Cool projects that will push your skills to the limit

Yii Rapid Application Development

Become a RAD hotshot with Yii, the world's most popular PHP framework

HOTSH🎯T

Lauren J. O'Meara
James R. Hamilton III

[PACKT] open source*
community experience distilled
PUBLISHING

Yii Rapid Application Development Hotshot

Become a RAD hotshot with Yii, the world's most popular PHP framework

Lauren J. O'Meara

James R. Hamilton III

Yii Rapid Application Development Hotshot

Credits

Authors

Lauren J. O'Meara

James R. Hamilton III

Reviewers

Thomas Jantz

Alexander Makarov

SAKURAI, atsushi (mocapapa)

Acquisition Editor

Robin de Jongh

Lead Technical Editor

Azharuddin Sheikh

Technical Editor

Veronica Fernandes

Project Coordinator

Abhishek Kori

Proofreader

Maria Gould

Indexer

Tejal R. Soni

Production Coordinator

Nitesh Thakur

Cover Work

Nitesh Thakur

About the Authors

Lauren J. O'Meara was lured into the field of computing from being a Mathematics major when she took Jeff Ondich's CS117 Computer Science course at Carleton College in Northfield Minnesota. After completing her degree in Computer Science, she wrote computer security software on both coasts of the US, at the MITRE corporation, and then at OneSecure (later acquired by NetScreen, and then by Juniper Networks). After spending some time gaining a broader understanding of business in the Santa Clara University MBA program, she flung herself into the world of web application development.

Today her web and mobile development consulting company, Plum Flower Software, is based out of Music City, USA.

I would like to thank Jeff Ondich and Mike Tie for getting me into computers; Todd O'Boyle for his sage technical and career advice; Randy Bias for being an entrepreneurial model; my brother, Thomas Jantz for his meticulous testing and feedback, and support; my father, mother, and sisters Tom, Nancy, Jennifer, and Kristin Jantz for their encouragement; my co-author, James R. Hamilton III, for his positivity, enthusiasm, and support; and the Yii developers, contributors, and community for sheer awesomeness.

About the Reviewers

Thomas Jantz is a little brother, friend, and business associate of the authors. After a brief love affair with Computer Science in his younger days, Thomas spent several years sewing his wild oats in the country before returning to computers in the spring of 2012. He works and trains closely with Plum Flower Software and enjoys programming in PHP, Java, Perl, and JavaScript.

> I would like to thank the authors Lauren J. O'Meara and James R. Hamilton III of Plum Flower Software for the opportunity to be one of the first to review and work through *Yii Rapid Application Development Hotshot*, and Abhishek Kori of Packt Publishing for his pleasant correspondence during the process.

Alexander Makarov is an experienced engineer from Russia and has been a Yii framework core team member since 2010. He is the author of another Yii framework book *Yii 1.1 Application Development Cookbook, Packt Publishing*.

> Thanks to my family for having being patient with me for taking on yet another project and spending less time with them.
>
> Thanks to the Yii community for their encouragement and contributions. It would not be possible to get free time if we were creating Yii without all these pull-requests at github. You're the best!

SAKURAI, atsushi (mocapapa) is a microprocessor expert as well as a
PHP programmer with over 10 years' experience. As a manager of a design team
of microprocessors, he has been working to build the support website for their
microprocessors. Recently, his efficiency for developing web application is drastically
increasing thanks to Yii. He has worked on the the book, *Yii 1.1 Application Development
Cookbook*, *Packt Publishing*. His main contribution for the Yii community includes translation
of the documentations in Japanese. He is a founder of *Yiijan*, a Japanese Yii users group.

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.

`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.

Table of Contents

Preface

As web developers, we are always looking for new and better tools to help us develop quality websites. Yii caught our eye as a great framework. It is known for performance. In addition to its speed, Yii provides great tools and features to help you get your job done quickly.

In this book, we highlight some of these features and capabilities, and demonstrate a few of the myriad ways you can use Yii. We hope to provide a fun journey through a complete web project and a catalogue of some common web development problems with their solutions in Yii.

What this book covers

Project 1, *Develop a Comic Book Database*, helps you to set up your Yii development environment and create a project in Yii.

Project 2, *Turn That DB into a Personal Mobile App*, lets you to extend Yii to serve mobile content, using jQuery Mobile.

Project 3, *Access all Areas – Users and Logins*, teaches you to add users and user management to your site.

Project 4, *Level Up! Permission Levels*, teaches you to add and configure access control for different functions of your site.

Project 5, *Service Please – Integrating Service Data*, helps you to incorporate other information sources into your site.

Project 6, *It's All a Game*, lets you have fun with your data by using it to make games.

Project 7, *Let It Work While You Sleep - Reports and Job Queues*, helps you to add a job queue manager to your project and display the collected data in charts and graphs.

Project 8, *Extend Yourself – Make a Module for Reuse*, teaches you to make your code reusable by converting it into a module.

What you need for this book

This book assumes that you have some familiarity with the development system of your choice and some background in writing programs. However, a motivated novice can fill in any knowledge gaps with a little outside research.

The examples are given in the context of a Linux system. The first project in this book will show you how to find, download, install, and configure the software that you will need to work through the projects in the book.

Who this book is for

This book is for PHP developers who want to learn how to develop with Yii, and for Yii developers who want to expand their toolkit.

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:

Mission Briefing

This section explains what you will build, with a screenshot of the completed project.

Why Is It Awesome?

This section explains why the project is cool, unique, exciting, and interesting. It describes what advantage the project will give you.

Your Hotshot Objectives

This section explains the major tasks required to complete your project.

- ▸ Task 1
- ▸ Task 2
- ▸ Task 3
- ▸ Task 4, and so on

Mission Checklist

This section explains any pre-requisites for the project, such as resources or libraries that need to be downloaded, and so on.

Task 1

This section explains the task that you will perform.

Prepare for Lift Off

This section explains any preliminary work that you may need to do before beginning work on the task.

Engage Thrusters

This section lists the steps required in order to complete the task.

Objective Complete - Mini Debriefing

This section explains how the steps performed in the previous section allow us to complete the task. This section is mandatory.

Classified Intel

The extra information in this section is relevant to the task.

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: "Copy the `include` directory from the package into the root XAMPP directory."

A block of code is set as follows:

```
array(
  'label'=>'Comic Books',
  'url'=>array('/book'),
  'items' => array(
    array('label'=>'Publishers',
      'url'=>array('/publisher')),
  )
),
```

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

```
array('allow', // allow admin user to perform 'delete' actions
  'actions'=>array('delete'),
  'users'=>array('admin'),
),
```

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

```
cd ~
mkdir projects
cd projects
```

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: "Click on **Install** and follow the installation instructions."

> 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/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 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.

Project 1
Develop a Comic Book Database
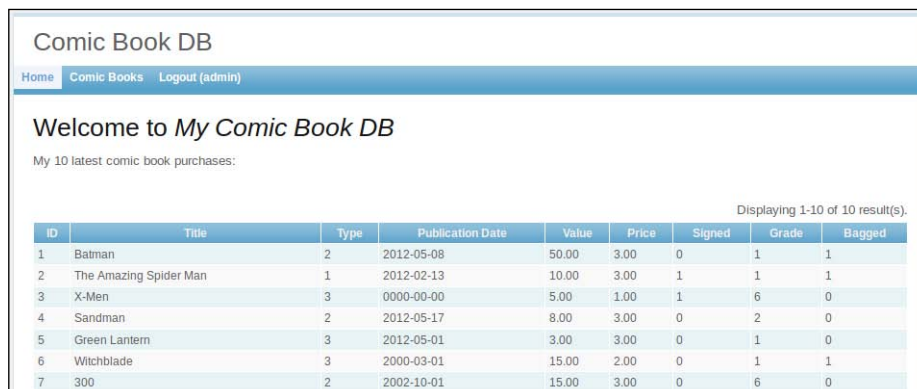
In this first mission, we will build a personal comic book database with input, update, list, delete, and search capabilities. In the process, we will set up a slick development environment, review Yii basics, and learn some handy tricks and shortcuts.

Mission Briefing

The purpose of this project is to introduce you to Yii and to some great development tools. If you are already familiar with Yii basics, you still might want to check out the tools in this project. NetBeans offers many convenient tools for a PHP developer, such as:

- ▶ Integrated unit testing with PHPUnit
- ▶ Convenient debugging with Xdebug
- ▶ Embedded terminal access

If you are a developer who does not enjoy systems administration tasks, XAMPP provides a convenient way to get your development stack up and running quickly. By the end of this project, you will have set up your development environment, created, scaffolded, and customized a Yii project, and developed and ran some tests against your code. You will have a web app capable of cataloging your comic book collection, and the home page of the site will look something like this (minus any customizations you choose to add):

Why Is It Awesome?

If you have been a comic book collector for some length of time, your collection has probably sprawled beyond easy memory access. When you come across an interesting issue, it can be hard to remember—Do I already have this one? A database will help catalogue and organize the items that you have, and it can be extended to keep a list of the items that you want. Yii provides a rapid application development framework that enables us to create this functionality in minutes and hours rather than days. This project can be easily adapted to any other type of item that you might collect.

Your Hotshot Objectives

- Setting up the LAMP Stack in One Step with XAMPP
- Installing NetBeans IDE
- Adding Xdebug to the Tool Set
- Unpacking the Yii Framework
- Initializing the Application Database
- Generating an Application Scaffold
- Beginning to Customize the App
- Getting Familiar with NetBeans and PHPUnit Testing Tools

Mission Checklist

We are going to be setting up our project on 32-bit Ubuntu 12.04. All of these tools are available for Windows and Mac. If you have a Mac, Windows, or a 64-bit system and would like to follow exactly, you could set up a virtual machine with Ubuntu and work from there. Another alternative, especially if you have a background in systems administration, is to set up each individual service in the LAMP stack:

- ▶ Apache
- ▶ MySQL
- ▶ PHP

If you go this route, the instructions will no longer be exact, but they will serve as guidelines and checklists for additional tools you can install.

Setting up the LAMP Stack in One Step with XAMPP
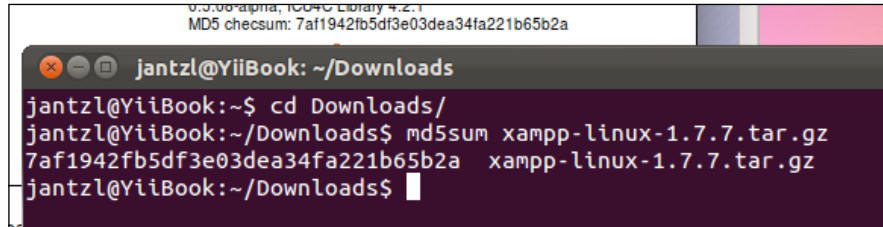
We are going to need a web server, a database, and PHP in order to write this application. LAMP is a term describing a software bundle that includes all of these pieces. XAMPP is one such package. It enables us to install our development tools in one shot. You can achieve the same effect by installing each piece yourself or by using an alternate LAMP package, such as WAMP, for Windows, or MAMP, for Mac OS.

Engage Thrusters

1. Go to `http://www.apachefriends.org/en/xampp.html` and find **XAMPP for Linux**.

2. Download the latest version of XAMPP and the development package.

3. Compare the md5 checksums on your system against the md5 checksums on the XAMPP download page to verify the packages by opening a terminal window and entering the following command:

   ```
   cd ~/Downloads
   md5sum xampp-linux-1.7.7.tar.gz
   md5sum xampp-linux-devel-1.7.7.tar.gz
   ```

After running the preceding command, this is what you will see:
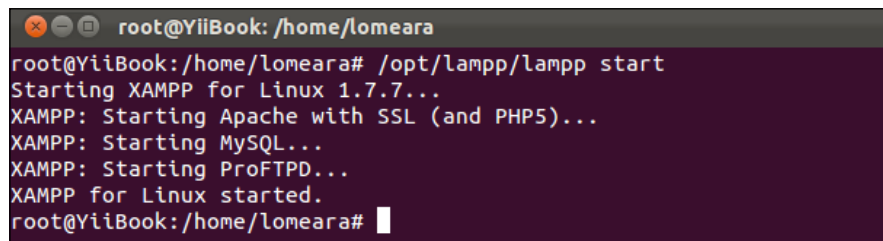
4. Using full system permissions, unpack the XAMPP package into a public directory.

```
sudo su

tar xzvf xampp-linux-1.7.7.tar.gz –C /opt
```

5. Start XAMPP.

```
/opt/lampp/lampp start
```

You should see something similar to the following screenshot:

6. Test your installation by firing up a browser and viewing `localhost` (`http://localhost`).

Objective Complete-Mini Debriefing

In one shot, you have installed your LAMP development stack: PHP, MySQL, and Apache, as well as some complimentary tools such as **webalizer**, **phpmyadmin**, **openssl**, and **pear**. Configuration, data, and logs live under one directory, that is where you installed XAMPP, in our example `/opt/lampp`.

Classified Intel

Please note that the XAMPP package is recommended for development. If you are deploying your work to a public server, you should research a proper security configuration for your system. We did not do anything with the XAMPP development package, but we will use it later, in the *Adding Xdebug to the Tool Set* task. One thing you will want to do right away is set up XAMPP to start on reboot. The installation package does not do this for you on Ubuntu. Here is how you do it:

1. As root, create an init script named `lampp` in `/etc/init.d` with the following contents:

   ```
   #!/bin/bash
   /opt/lampp/lampp start
   ```

2. Make the file executable as follows:

   ```
   sudo chmod +x lampp
   ```

3. Use `update-rc.d` to install this init script at all run levels.

   ```
   sudo update-rc.d lamp defaults
   ```

Now when you restart your server, XAMPP will start automatically.

Apache User Sharing

1. If the user `www-data` does not already exist (check `/etc/passwd` for an entry for `www-data`), create it.

   ```
   sudo adduser --system --group --no-create-home www-data
   -quiet
   ```

2. Add your user to the `www-data` group by editing `/etc/group` and adding yourself to the `www-data` line. For example, the following line adds the user named `lomeara` to the `www-data` group:

   ```
   www-data:x:33:www-data,lomeara
   ```

3. As root, edit the Apache configuration file `/opt/lampp/etc/httpd.conf`:

   ```
   sudo gedit /opt/lampp/etc/httpd.conf
   ```

4. Change the entry for `User` and `Group` to `www-data`.

   ```
   User www-data
   Group www-data
   ```

5. Restart XAMPP.

   ```
   sudo /opt/lampp/lampp restart
   ```

6. Confirm that XAMPP is running as `www-data`.

```
ps aux | grep lampp
```

Your output should include some lines that look like the following:

```
www-data  3402  0.0  1.2  50512 12844 ?       S    22:25
0:00 /opt/lampp/bin/httpd -k start -DSSL -DPHP5 -E
/opt/lampp/logs/error_log
```

Installing NetBeans IDE

In the last task, we installed the LAMP stack that will run our app. Now, we will begin to put together our development tool set, starting with an integrated development environment, NetBeans.

Engage Thrusters

1. Java 7 and JDK 7 are required for the current version of NetBeans (7.1.1). Install them as per your operating system.

   ```
   sudo apt-get install openjdk-7-jdk openjdk-7-jre
   ```

2. Download the NetBeans installer from `netbeans.org`. We downloaded the PHP bundle `netbeans-7.1.1-ml-php-linux.sh`.

3. If you wish to put NetBeans in a public location, run the installer as root. If you would like to install it for your workspace only, run it as yourself. We are going to install as ourselves.

   ```
   chmod +x netbeans-7.1.1-ml-php-linux.sh
   ```

   ```
   ./netbeans-7.1.1-ml-php-linux.sh
   ```

4. Run NetBeans.

5. On the NetBeans start page, click on **Install Plugins**.

6. Search for **Selenium**.

7. Check the box for **Selenium Module for PHP** in the search results.

8. Click on **Install** and follow the installation instructions.

9. Use the following commands to update Pear. In case you have another instance of PHP/Pear on your system, be sure to work with the one under XAMPP.

   ```
   sudo /opt/lampp/bin/pear channel-discover pear.phpunit.de
   ```

   ```
   sudo /opt/lampp/bin/pear channel-discover components.ez.no
   ```

   ```
   sudo /opt/lampp/bin/pear channel-discover pear.symfony-
   project.com
   ```

10. Install PHPUnit.

```
sudo /opt/lampp/bin/pear install phpunit/PHPUnit

Install Selenium integration.

sudo /opt/lampp/bin/pear install phpunit/PHPUnit_Selenium
```

11. Install Story-based test runner for behavior-driven development.

```
sudo /opt/lampp/bin/pear install phpunit/PHPUnit_Story
```

12. Configure NetBeans to use PHPUnit. Open **Tools** | **Options** | **PHP** | **Unit Testing** and set the correct path: `/opt/lampp/bin/phpunit`.

Objective Complete-Mini Debriefing

You have installed NetBeans for PHP which will, of course, give you a code editor and project navigation. Some of the things we love include quick access to unit testing, debugging (once we have set up Xdebug), and code coverage. It will also provide remote access to your app once you have deployed it.

Classified Intel

NetBeans provides framework support for Zend, Symfony, and Smarty. It does not currently have support for Yii, but with some configuration tweaks, we will have the benefits of completion and search. Yii provides all the framework tools we will need.

Adding Xdebug to the Tool Set

Continuing to gather and prepare our development tools, we will install Xdebug under XAMPP where it will be accessible from NetBeans.

Engage Thrusters

1. Unpack the XAMPP development package.

```
cd ~/Downloads

tar xzvf xampp-linux-devel-1.7.7.tar.gz
```

2. Copy the `include` directory from the package into the root XAMPP directory.

```
sudo cp -r lampp/include /opt/lampp/.
```

3. Update the PECL channels database. Remember to work with the instance of PECL under XAMPP if you have more than one instance on your system.

```
sudo /opt/lampp/bin/pecl update-channels
```

13

4. The `autoconf` package is a prerequisite for Xdebug. Use the following command to find out if it is installed:

`sudo dpkg --list autoconf`

If `autoconf` is installed, the output will look like the following:

```
||/ Name           Version         Description
+++-=============-===============-
=========================================
ii  autoconf      2.68-1ubuntu2  automatic configure script
builder
```

If `autoconf` is not installed, the output will look like the following:

`No packages found matching autoconf`

Use the following command to install `autoconf`.

`sudo apt-get install autoconf`

5. Install Xdebug with PECL.

`sudo /opt/lampp/bin/pecl install Xdebug`

6. The output of the `pecl` command will confirm the installation of Xdebug in XAMPP extensions. Edit the XAMPP `php.ini` file.

`sudo gedit /opt/lampp/etc/php.ini`

Add a line to the end of the file to include the Xdebug extension:

`zend_extension = "/opt/lampp/lib/php/extensions/no-debug-non-zts-20090626/xdebug.so"`

Also add lines to configure Xdebug:

`xdebug.remote_enable = 1`

`xdebug.remote_handler = "dbgp"`

`xdebug.remote_host = "localhost" xdebug.remote_port = 9000`

7. Restart XAMPP.

8. Confirm that the extension is activated by visiting the `phpinfo()` page: `http://localhost/xampp/phpinfo.php`.

> This program makes use of the Zend Scripting Language Engine:
> Zend Engine v2.3.0, Copyright (c) 1998-2011 Zend Technologies
>   with Xdebug v2.2.0rc1, Copyright (c) 2002-2012, by Derick Rethans
>
> Powered By
> Zend Engine 2
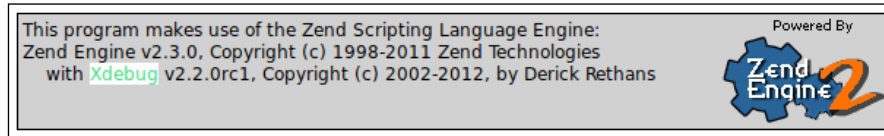
Objective Complete - Mini Debriefing

Now you can set breakpoints in your code to pause execution, step through the lines of code, and examine the values of the variables.

Classified Intel

Once you have hit a breakpoint, you can view the contents of the variables in the toolbars, or when you move the mouse over a variable in the code viewer, the value will appear in the tooltip.

Unpacking the Yii Framework

Once we add one final tool, Yii, we will initialize our project and load it into NetBeans. Adding some Yii-specific configuration to our project in NetBeans will expand the array of tools at hand to include Yii and PHPUnit.

Engage Thrusters

Before we go anywhere, we must grab a copy of Yii and place it where XAMPP can access it.

Installing Yii

1. Download the latest version of Yii from `http://www.yiiframework.com/` (currently 1.1.10).

2. Unpack the Yii tarball.

   ```
   cd ~/Downloads

   tar xzvf yii-1.1.10.r3566.tar.gz
   ```

3. Move the directory into the XAMPP root.

   ```
   sudo mv yii-1.1.10.r3566 /opt/lampp/htdocs/.
   ```

4. Create a symbolic link from the version-named Yii directory, `yii-1.1.10` in our example, to a directory named `yii`. (This step is not necessary, but it can be useful when we upgrade. If all outside references use Yii, then we can just change the symbolic link when we upgrade Yii.)

   ```
   cd /opt/lampp/htdocs/
   sudo ln -s yii-1.1.10.r3566/ yii
   ```

5. Check your Yii installation by visiting `http://localhost/yii/requirements/`. You may see warnings for Memcache and APC extensions. These extensions are caching utilities for optimizing your site. You can develop without them.
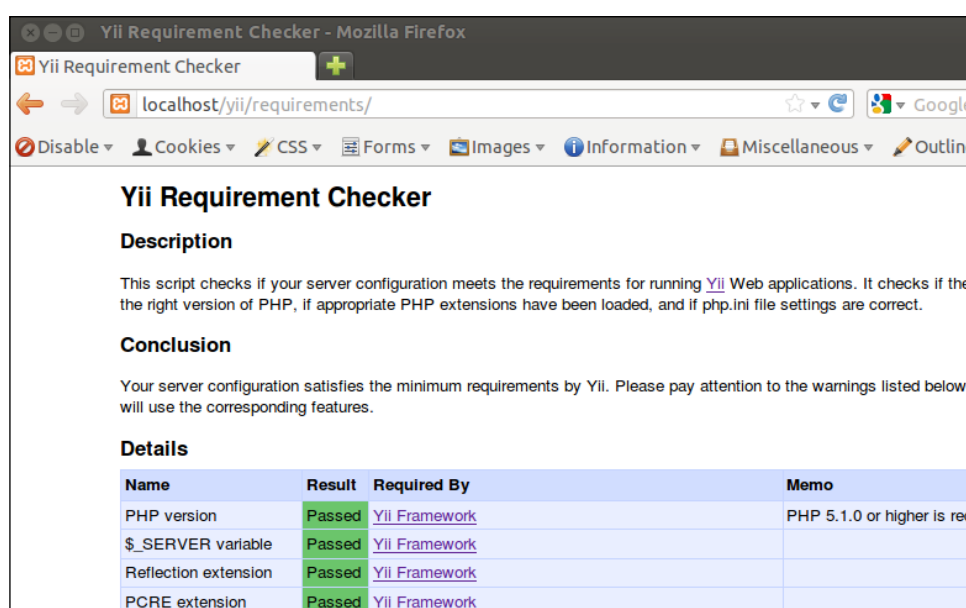
6. Add XAMPP and Yii framework directories to your executable path. For example, if you are working on a Unix-based system and use bash in your terminal, you can do this by adding the following line to `.bashrc` in your home directory.

   ```
   export PATH=${PATH}:/opt/lampp/bin:/opt/lampp/htdocs/yii/framework
   ```

7. Use the source command to evaluate `.bashrc` in any open terminal windows to pick up the changes.

   ```
   source ~/.bashrc
   ```

Creating a Yii project

1. Select a directory for your project. We create a directory named `projects` in our home directory.

   `cd ~`

   `mkdir projects`

   `cd projects`

2. Run the `yiic` command to create scaffolding for our project.

   `yiic webapp cbdb`

3. You will be asked if you want to **Create a web application under /home/lomeara/ projects/cbdb?**, and you will be provided with two options, **[yes|no]**. Select **yes**.

4. Create a link to our new `webapp` directory in the XAMPP `webroot` directory.

   `sudo ln –s ~/projects/cbdb /opt/lampp/htdocs/.`

5. Change ownership of the directories in your project that are subject to code generation.

   `cd ~/projects/cbdb/protected`

   `sudo chgrp www-data models controllers views`

6. Open `http://localhost/cbdb/` in your web browser to view your newly created project.
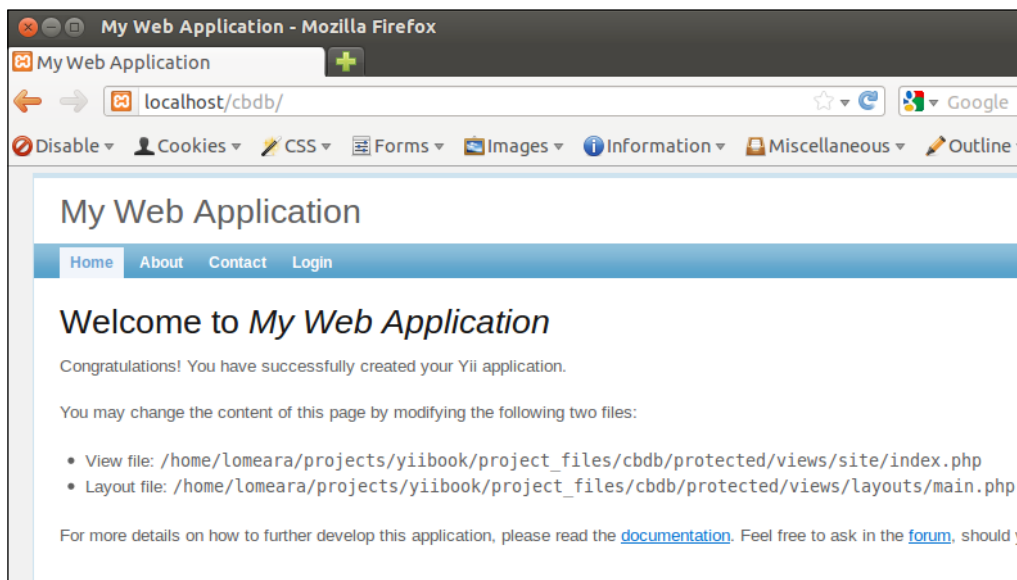
Adding project in NetBeans

1. In a terminal window, determine the version of PHP that you are running under XAMPP.

    ```
    /opt/lampp/bin/php -v
    ```

    You will see output that looks like the following:

    ```
    PHP 5.3.8 (cli) (built: Sep 19 2011 13:29:27)
    Copyright (c) 1997-2011 The PHP Group
    Zend Engine v2.3.0, Copyright (c) 1998-2011 Zend
    Technologies
        with Xdebug v2.2.0rc1, Copyright (c) 2002-2012, by Derick
    Rethans
    ```

2. Open NetBeans and click on **File** | **New Project**:
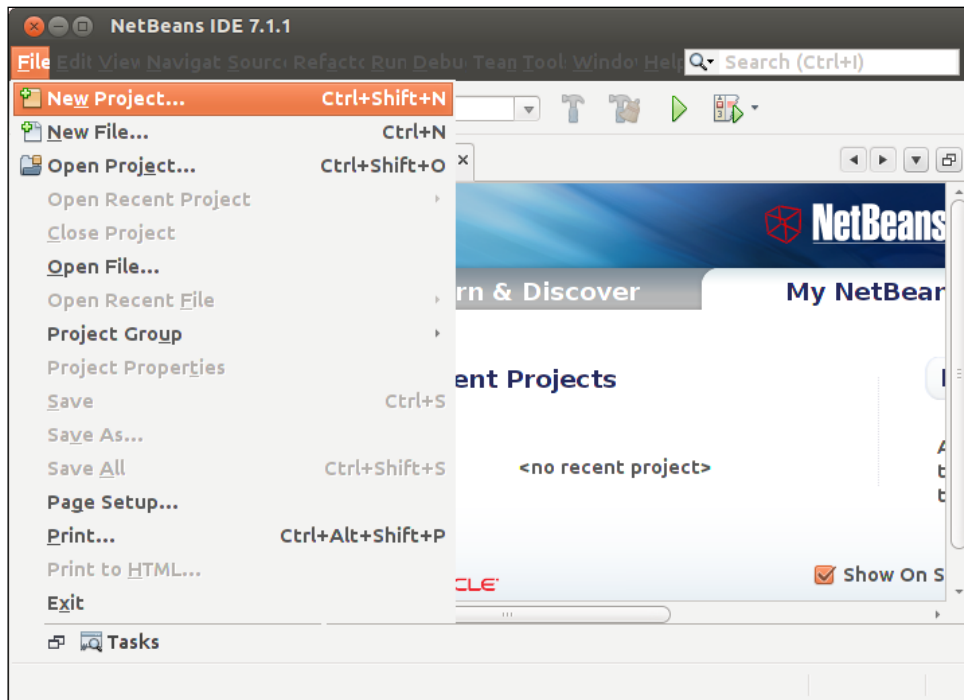
3. Select **PHP Application with Existing Source** and click on **Next**.
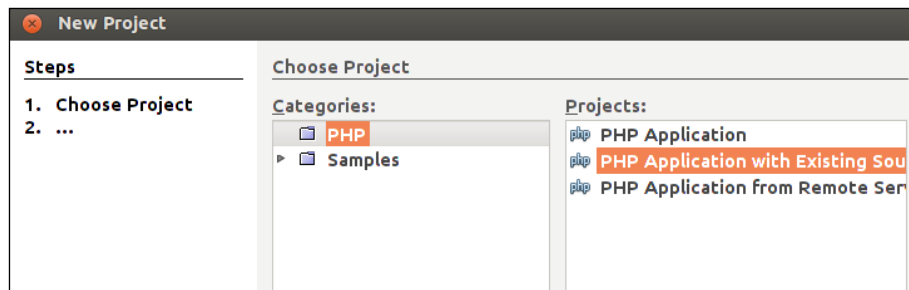
4. Browse to your project directory (For example, `/home/lomeara/projects/cbdb`).

5. Select the correct version of PHP from step 1 (for us, the version is 5.3) and click on **Next**.
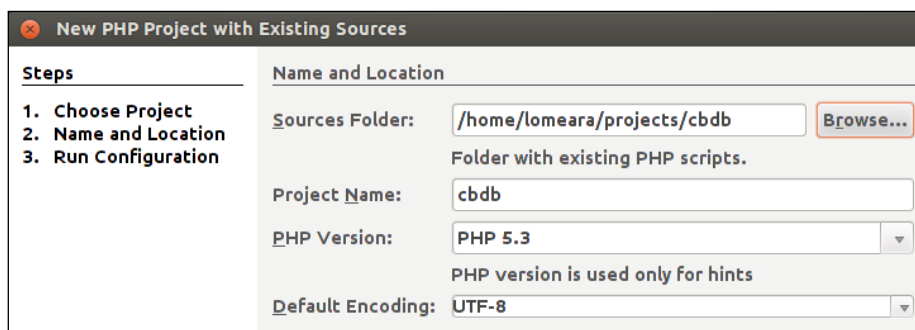
6. Accept the defaults on the next screen and click on **Finish**. You should now see **cbdb** in the project window.
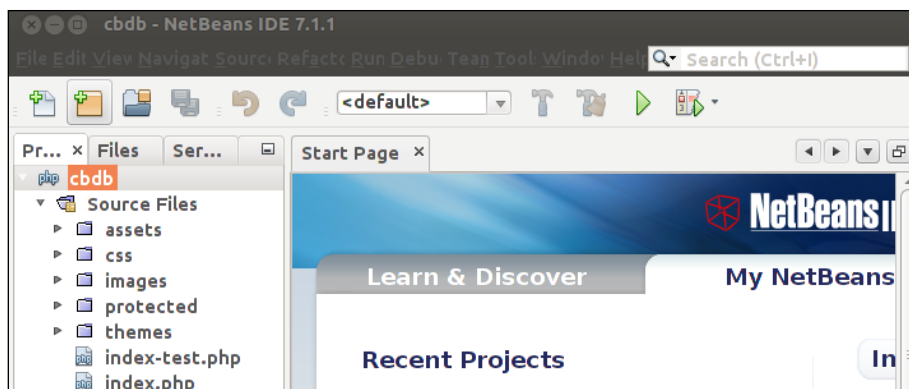
7. Right-click on the project name and select **Run** to launch the webapp in a browser.
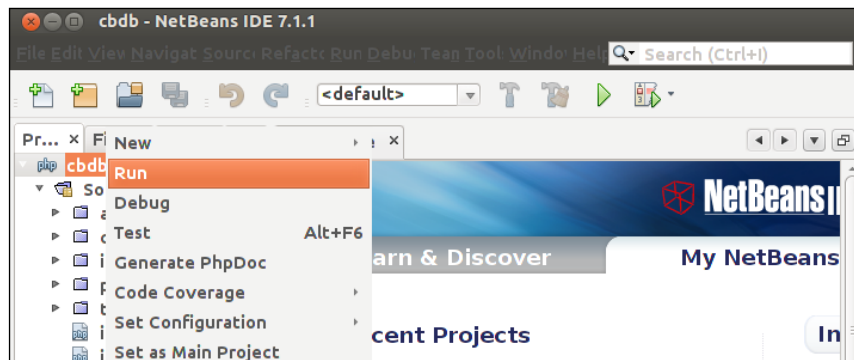
Configuring the project in NetBeans

1. Click on **File | Project Properties**.

2. In the Sources category, select the test folder (`/home/lomeara/projects/cbdb/protected/tests`).

3. In the PHPUnit category, activate **Use Bootstrap**, and select the Bootstrap file (`/home/lomeara/projects/cbdb/protected/tests/bootstrap.php`). Then activate **Use XML Configuration** and select the XML configuration file (`/home/lomeara/projects/cbdb/protected/tests/phpunit.xml`).

4. In the PHP Include Path category, add the Yii Framework root (`/opt/lampp/htdocs/yii`) and PHPUnit (`/opt/lampp/lib/php/PHPUnit`).

5. Open **Tools | Options | Miscellaneous | Files** and in the field **Files Ignored by the IDE** add `yiilite\.php|` in front of `CVS`. The result will look something like the following screenshot:
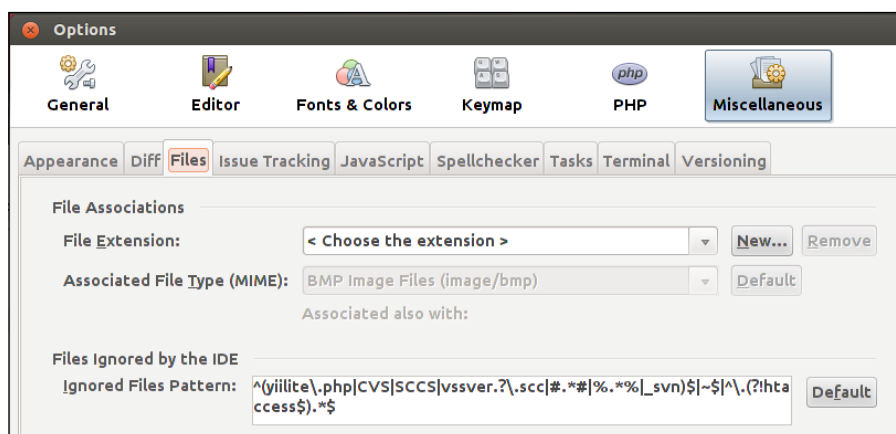
6. Open **Tools** | **Options** | **PHP** | **Debugging** and uncheck **Stop at First Line**.

7. Restart NetBeans.

Objective Complete - Mini Debriefing

We have created our first project and added it to NetBeans. We have updated the NetBeans project to utilize PHPUnit. There are just a few more useful tools to put at our fingertips as we move forward.

Initializing the Application Database

We are going to set up a database connection in NetBeans and load in a basic schema to get you started.

Prepare for Lift Off

We have put together some tables to capture basic comic book information:

- ▶ `book`
- ▶ `author`
- ▶ `illustrator`
- ▶ `type` (serial, trade, graphic novel)
- ▶ `grade` (mint, near mint, fair, poor, and so on)

Yii requires consistency in table naming: singular or plural, but not both. Singular is recommended for simplicity.

Engage Thrusters

1. In NetBeans, go to **Windows** | **Services**.

2. Right-click on **Databases** and select **New Connection…**.

3. You should see a MySQL driver. Click on **Next**.

4. Accept the connection defaults. (Click on the **Test Connection** button to verify it works).

5. Click on **Finish**.

6. Right-click on **MySQL Server** and select **Create Database…**.

7. In the **New Database Name:** field, enter `cbdb`.

8. Check **Grant Full Access To:** and select ***@localhost**.

9. Click on **Ok**.

10. Right-click on **jdbc:mysql://localhost:3306/cbdb** and select **Execute Command**.

11. Paste the contents of `schema.sql` into the editor window (labeled **SQL Command 1**).

> **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.

12. Click on the **Run SQL** button (or *Ctrl* + *Shift* + *E*) to create the tables.

This entity relationship diagram illustrates the relations between the tables:
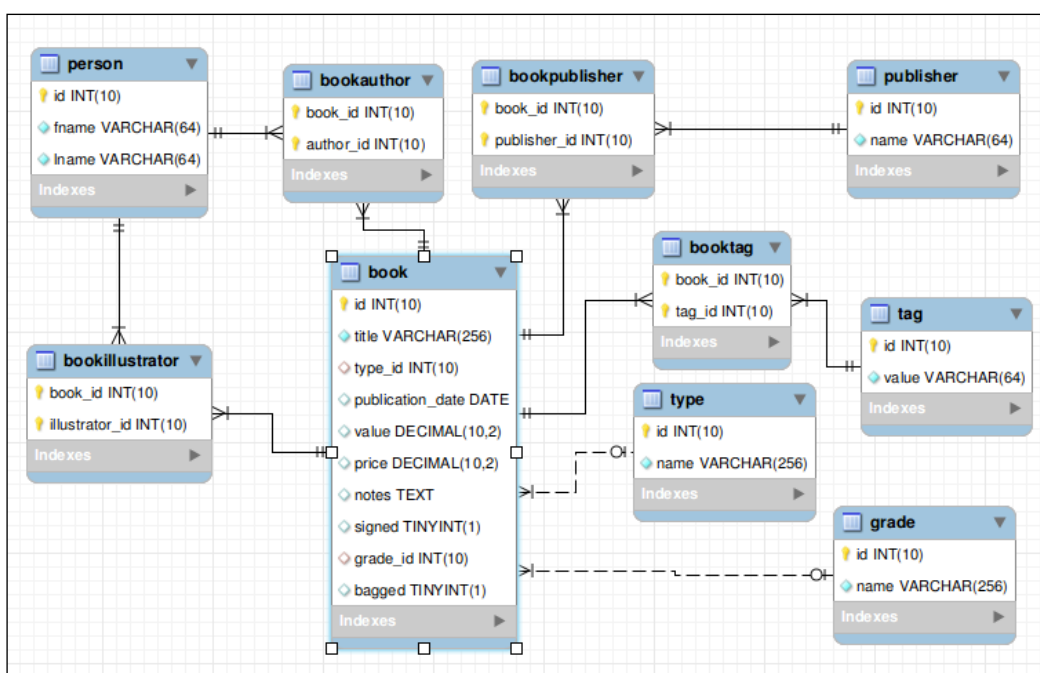
Objective Complete - Mini Debriefing

Now that the schema has been loaded, we can use the IDE to inspect the tables and contents. To do so, click on **cbdb** in the **Services** window to expand it and see the table fields. As you can see, a book record includes an ID, title, type, publication date, value, price, grade, notes, and flags for signed and bagged. This table contains no data, so let's look at a table that does have information in it. Right-click on **grade** and select **View Data**. A SQL command window will open with the query in it. Below that, you should see the output of the query, the data that is in the grade table.
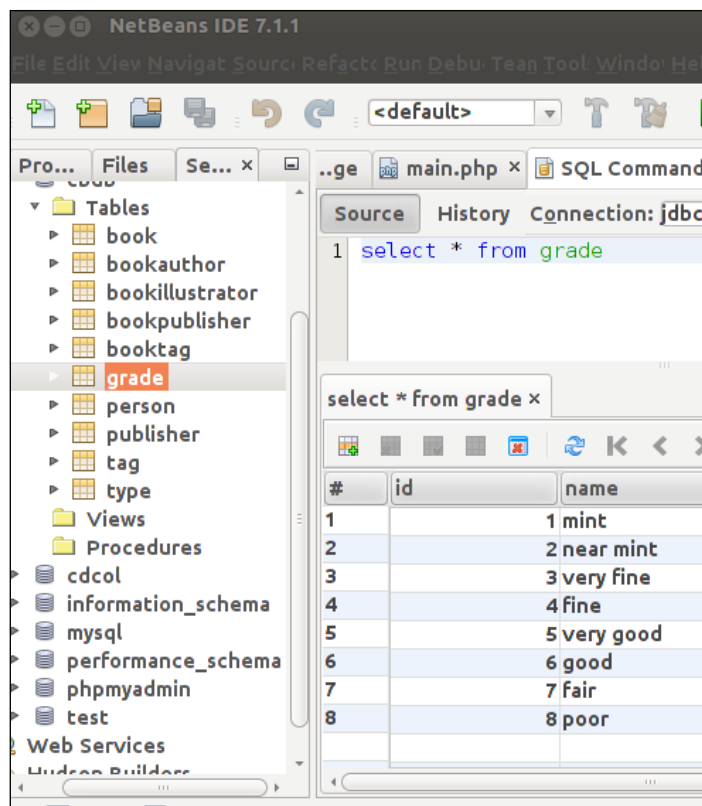
NetBeans provides a nice interface for interacting with a MySQL database. As we have demonstrated, you can access and inspect the database as well as build queries and execute commands. This is useful when you are working with one database and even better when you are working with more than one.

Generating an Application Scaffold

Yii provides some great web-based scaffolding tools. Now that we have a schema in place, we can use those tools to provide basic access to our comic book objects.

Prepare for Lift Off

First, we have to update our application configuration to point to our database. Then we will enable **Gii**, Yii's graphical code generator. With Gii, we will generate scaffolding for each entity in our database. At the end, we will have a very basic, but functional, comic book database webapp.

Engage Thrusters

1. Fire up NetBeans and go to the **Projects** window.

2. Expand **cbdb** | **Source Files** | **protected** | **config**.

Updating Database Configuration

1. Open `main.php`.

2. Scroll down to the **db** section under **components**. You will see an active entry for the default sqlite database.

3. Let's start by commenting out that entry to disable it.

```
/*
    'db'=>array(
      'connectionString' => 'sqlite:'.dirname(__FILE__).'/../data/
       testdrive.db',
    ),
*/
```

4. Now uncomment the example MySQL entry below that and input our database information. Change the dbname to cbdb. Everything else should be the same, unless you created a user and/or set a password for your database.

```
    'db'=>array(
      'connectionString' =>
        'mysql:host=localhost;dbname=cbdb',
      'emulatePrepare' => true,
      'username' => 'root',
      'password' => '',
      'charset' => 'utf8',
    )
```

Enabling Gii

1. We are going to enable Gii so that we can scaffold the objects that are in our database. Scroll up to the `modules` section of the config file (around line 23).

2. Uncomment the `gii` section and give the `password` attribute a value.

```
    'modules'=>array(
      'gii'=>array(
        'class'=>'system.gii.GiiModule',
        'password'=>'yiibook',
        // If removed, Gii defaults to localhost only. Edit
        //carefully to taste.
        'ipFilters'=>array('127.0.0.1','::1'),
      )
```

3. Right-click on the project and select **Run** to go to the webapp in a browser, then add `?r=gii` to the URL. The whole URL should be `http://localhost/cbdb/index.php?r=gii`. Hit **Enter** to load Gii.

4. Enter the password that you specified in the config file.

5. Select **Model Generator** to get started; we are going to generate models for all of the tables in the database.

6. We will start with a table that does not have dependencies—`type`. Enter `type` in the **Table Name** field. You will see that the field **Model Class** is filled in for you as you type. The screen should look like this:

Generators	Model Generator
Controller Generator	This generator generates a model class fo...
Crud Generator	
Form Generator	*Fields with * are required. Click on the* hig...
Model Generator	**Table Prefix**
Module Generator	[empty]
	Table Name *
	type
	Model Class *
	Type
	Base Class *
	CActiveRecord
	Model Path *
	application.models
	Build Relations
	☑
	Code Template *
	default (/opt/lampp/htdocs/yii-1.1.10.r3566
	Preview

7. Click on the **Preview** button to review the code that will be generated. Then click on **Generate**.

8. Repeat steps 6 and 7 for each table in the following order: `grade`, `book`, `publisher`, `bookpublisher` (change the suggested model name to `BookPublisher`), `person`, `bookauthor` (change the suggested model name to `BookAuthor`), `bookillustrator` (change the suggested model name to `BookIllustrator`), `tag`, and `booktag` (change the suggested model to name `BookTag`).

9. Now we will create views and controllers for all of the models we just created. Click on **Crud Generator** in the **Generators** menu.
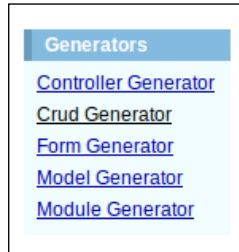
10. Enter `Type` in the **Model Class** field. Now you will see the **Controller ID** field auto-completed for you.

11. Click on **Preview** and you will see a list of many files: one controller and several view items.

Code File	Generate ☑
controllers/TypeController.php	new ☑
views/type/admin.php	new ☑
views/type/create.php	new ☑
views/type/_form.php	new ☑
views/type/index.php	new ☑
views/type/_search.php	new ☑
views/type/update.php	new ☑
views/type/_view.php	new ☑
views/type/view.php	new ☑

Preview   Generate

12. Click on **Generate** to create the files. The output will include a link to **try it now**. You can click on this to open a new window and test out the CRUD functionality you just created for the `Type` object.

13. Repeat steps 10, 11, and 12 for each model: Grade, Book, Publisher, Person, Tag. We are not going to generate CRUD for our join tables.

Objective Complete - Mini Debriefing

Great; we have a functioning web app now. Sure, it is not ready for prime time. We are going to want to spend some time making the interface user-friendly and connecting the objects. But it's not a bad start.

If you go back to the main screen, you will not see any links to get you to the objects we just added. For the moment, you can get to them by manipulating the URL. To get to the books, change your URL to `http://localhost/cbdb/index.php?r=book`.
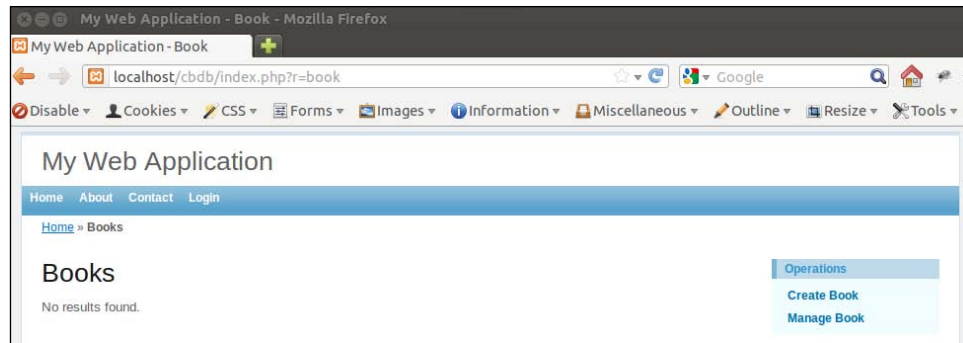
From this page, you can click on **Create Book** to add a book to our database (if you are asked to log in, use admin/admin). This interface leaves something to be desired. For one, we will have to input raw index and Boolean values in **Type**, **Signed**, **Grade**, and **Bagged**. If you input something like this, you can add an entry.

In the next section, we will connect these pieces to make the user experience much smoother.

Classified Intel

At this point, you can deactivate Gii. You get the picture about how powerful it is. You do not want to accidentally leave it activated when you deploy to a public server. So go back into the configuration file and comment out the Gii section. We can always uncomment it in the future if we need it again.

Beginning to Customize the App

Now we have a functional, but not very accessible site. We are going to start customizing the framework that Yii provides. This is where the real development work begins. In this project, we will concentrate on making what is available more accessible. You may want to frequently go back to the site in your web browser and reload to view the effects of these steps.

Engage Thrusters

Let's start by updating the menus so that we can click to our comic book information.

Menus

1. In NetBeans, expand **Source Files** | **protected** | **views** | **layouts** and open `main.php`.

2. This file is the main wrapper for your site. It contains the header and footer that are on every webpage.

3. For now, our site is for personal use, so we will leave the login and comment out `About` and `Contact` lines 32 and 33.

```
'items'=>array(
  array('label'=>'Home','url'=>array('/site/index')),
//array('label'=>'About', 'url'=>array
  ('/site/page', 'view'=>'about')),
//array('label'=>'Contact', 'url'=>array
  ('/site/contact')),
array('label'=>'Login', 'url'=>array('/site/login')),
 ),
```

4. In the line above `Login` (34), add new lines for the objects that we want to manage through the interface.

```
array('label'=>'Comic Books','url'=>array('/book')),
```

5. It would be nice to access the other items in the same way, but we don't want to fill up the screen with objects. Let's add the other objects to a drop-down list, under `Comic Books`, by creating a sublist containing a link to `Publishers`.

```
array(
    'label'=>'Comic Books',
    'url'=>array('/book'),
    'items' => array(
      array('label'=>'Publishers',
        'url'=>array('/publisher')),
    )
),
```

6. Now we will change the CSS for the site to display the drop-down menu. Open **Source Files | css | main.css** and replace the `mainmenu` section (around line 50) which looks like this:

```
#mainmenu
{
  background:white url(bg.gif) repeat-x left top;
}

#mainmenu ul
{
  padding:6px 20px 5px 20px;
 margin:0px;
}

#mainmenu ul li
{
  display: inline;
}

#mainmenu ul li a
{
  color:#ffffff;
  background-color:transparent;
  font-size:12px;
  font-weight:bold;
  text-decoration:none;
  padding:5px 8px;
}

#mainmenu ul li a:hover, #mainmenu ul li.active a
{
  color: #6399cd;
```

```
      background-color:#EFF4FA;
      text-decoration:none;
}
```

Replace the previous code with the following:

```
#mainmenu
{
  background:white url(bg.gif) repeat-x left top;
}

#mainmenu ul
{
  padding:6px 20px 5px 20px;
        list-style: none;
        margin: 0;
        padding: 0;
        position: relative;
        height: 30px;
}

#mainmenu ul li
{
  display: block;
        height: 28px;
        float: left;
        overflow: visible;
        position: relative;

}

#mainmenu li ul
{
        position: absolute;
        top: 24px;
        left: 10px;
}

#mainmenu li ul li
{
        display: none;
}

#mainmenu ul li a
{
```

```css
        float: left;
        display: block;
  color:#ffffff;
  background-color:transparent;
  font-size:12px;
  font-weight:bold;
  text-decoration:none;
  padding:5px 8px;
}

#mainmenu .active a, #mainmenu li:hover > a,
#mainmenu li:hover > ul li
{
  display: block;
  color: #6399cd;
  background-color:#EFF4FA;
  text-decoration:none;
}
```

7. If we are going to support login, we should move our protected menu item inside of the login. We could do this with the "visible" flag, but that will complicate things for our drop-down visibility. We chose to split the menu into a Logged In Version and a Not Logged In Version. Here is the result in **Source Files | protected | views | layouts | main.php**:

```php
<div id="mainmenu">
<?php
  if (Yii::app()->user->isGuest) {
    $this->widget('zii.widgets.CMenu',array(
      'activeCssClass' => 'active',
      'activateParents' => true,
      'items'=>array(
        array('label'=>'Home',
          'url'=>array('/site/index')),
      //array('label'=>'About',
        'url'=>array('/site/page', 'view'=>'about')),
      //array('label'=>'Contact',
        'url'=>array('/site/contact')),
      array('label'=>'Login',
        'url'=>array('/site/login')),
      ),
    ));
  } else {
    $this->widget('zii.widgets.CMenu',array(
        'activeCssClass' => 'active',
```

```
        'activateParents' => true,
        'items'=>array(
          array('label'=>'Home',
            'url'=>array('/site/index')),
          array('label'=>'Comic Books',
            'url'=>array('/book'),
           'items' => array(
             array('label'=>'Publishers',
               'url'=>array('/publisher')),
           )
          ),
          array('label'=>'Logout ('.Yii::app()->user-
            >name.')', 'url'=>array('/site/logout'))
        ),
      ));
    }
  ?>
  </div><!-- mainmenu -->
```

Forms

Let's make that comic book create form better. We will start with some quick-to-implement customizations. The first will be to fix the drop-down menus for Type and Grade.

1. Start by extending the Type model. Open **Source Files | protected | models | Type. php** and add a function that returns all of the Type values in a list format.

    ```
    public function getTypeOptions()
    {
            return CHtml::listData(Type::model()-
              >findAll(),'id','name');
    }
    ```

2. Then, change the book form (which conveniently happens to be the same file for create and update). Open **Source Files | protected | views | book | _form.php**. We will replace the section for Type which looks like the following:

    ```
    <div class="row">
            <?php echo $form-
              >labelEx($model,'type_id'); ?>
            <?php echo $form-
              >textField($model,'type_id',array
              ('size'=>10,'maxlength'=>10)); ?>
            <?php echo $form-
              >error($model,'type_id'); ?>
    </div>
    ```

33

The previous section should be changed from a text field to a dropdown, using our new model function, as follows:

```
<div class="row">
        <?php echo $form-
          >labelEx($model,'type_id'); ?>
        <?php echo $form->dropDownList($model,
          'type_id', Type::model()-
          >getTypeOptions()); ?>
        <?php echo $form-
          >error($model,'type_id'); ?>
</div>
```

3. Follow that format to make the **Grade** field a drop-down list too.

4. Similarly, let us change the input for the Boolean values **Signed** and **Bagged** by changing the text field to a checkbox. For example, we changed **Bagged** as follows:

```
<?php echo $form->checkbox($model, 'bagged'); ?>
```

After these changes, the create comic book form will look like the following screenshot:

5. We can make the publication date look very nice with the Yii Jui widget. Start by replacing the `publication_date` text field in the form with the following:

```php
<?php $this->widget('zii.widgets.jui.CJuiDatePicker',
  array(
  'name' => 'publication_date',
  'attribute' => 'publication_date',
  'model'=>$model,
  'options'=> array(
    'dateFormat' =>'yy-mm-dd',
    'altFormat' =>'yy-mm-dd',
    'changeMonth' => true,
    'changeYear' => true,
    'appendText' => 'yyyy-mm-dd',
  ),
));
?>
```


Objective Complete - Mini Debriefing

At this point, your site will include a custom menu that lets you click to access the comic book and publisher objects. It will also have a better, custom comic book create/update form with drop-down menus for **Type** and **Grade** and a date picker for **Publication Date**.

Classified Intel

You may have noticed that your URL contains a reference to the `index.php` file. You can configure Yii and Apache to provide cleaner, more readable URLs with the following changes:

1. In NetBeans, expand the **Source Files** | **protected** | **config** and open `main.php` to edit.

2. Change the name of the app from `My Web Application` to `Comic Book DB`.

   ```
   'name'=>'Comic Book DB'
   ```

3. Scroll down to `components` and uncomment the `urlManager` section.

4. Create a `.htaccess` file in the root directory of the project. (Right-click on **Source Files** | **New** | **Other**. Then select **Other** | **Empty File**. Give it the name `.htaccess` and click on **Finish**.)

   ```
   RewriteEngine on

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

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

5. Access `http://localhost/cbdb/book` in your web browser. You should see the book's index page.

Getting Familiar with NetBeans and PHPUnit Testing Tools

We advocate for Test Driven Design, but we are not going to review it in depth. Hopefully, you are already familiar with this methodology and aware of the enormous benefits it can impart to development. We are going to assume that you already know about test-driven design or will learn about it and bring it into your software development practice. In this task, we are just going to show you some tools you can use in its application.

Engage Thrusters

1. Currently, we have no tests set up, but we can get the hang of testing by right-clicking on the project root (`cbdb`) and selecting **Test**.

2. The tests (at this point, there are zero tests) will run and display results in the bottom pane—not very exciting. Let's make some tests. NetBeans will tell PHPUnit to generate tests for you. Expand **Source Files** | **protected** | **models**, right-click on **Type**, and select **Tools** | **Create PHPUnit tests**.
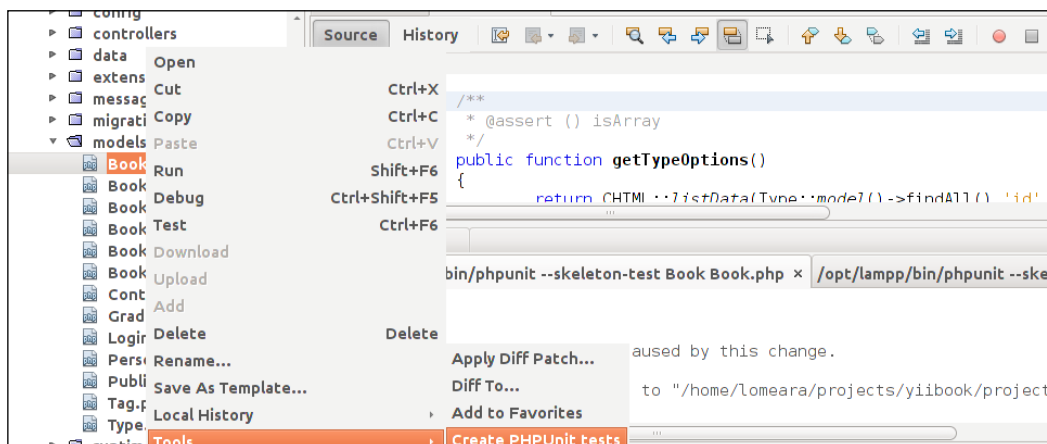
3. PHPUnit will create a file named `TypeTest.php` with tests for all of the functions in the Type model. Let's add an actual test for the `getTypeOptions` function we wrote earlier.

4. First, we need to create a test database. Go to the services window, right-click on your MySQL connection, and select **Create Database**.

5. Name the database `cbdb_test` and set **Grant Full Access To** to `*@localhost`.

6. Set up the tables by right-clicking on **Execute Command** for `cbdb_test` then inputting and running the exact same schema definition we used to create our development database.

7. Now we can configure the test database connection. Uncomment and complete the database section in **Source Files | protected | config | test.php**.

```
'db'=>array(
    'connectionString' => 'mysql:host=localhost;dbname=cbdb_test',
    'emulatePrepare' => true,
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
),
```

8. Next, we will add a test for the `getTypeOptions` function we wrote earlier. In `TypeTest.php`, scroll down to `testGetTypeOptions` (around line 100) and replace:

```
    // Remove the following lines when you implement
//this test
 $this->markTestIncomplete(
        'This test has not been implemented yet.'
    );
```

With the following:

```
    $options = Type::model()->getTypeOptions();
    $this->assertTrue(is_array($options));
    $this->assertEquals(3, count($options));
```

9. Let's prepare to run the test again, but this time collect code coverage information. First, we are going to tell PHPUnit that we only care about coverage for certain directories. Edit **Test Files | phpunit.xml** and add the following section:

```
<filter>
    <whitelist>
        <directory suffix=".php">../models</directory>
    </whitelist>
</filter>
```

10. Right-click on the project root (**cbdb**). Select **Code Coverage | Collect and Display Code Coverage** as shown in the following screenshot:
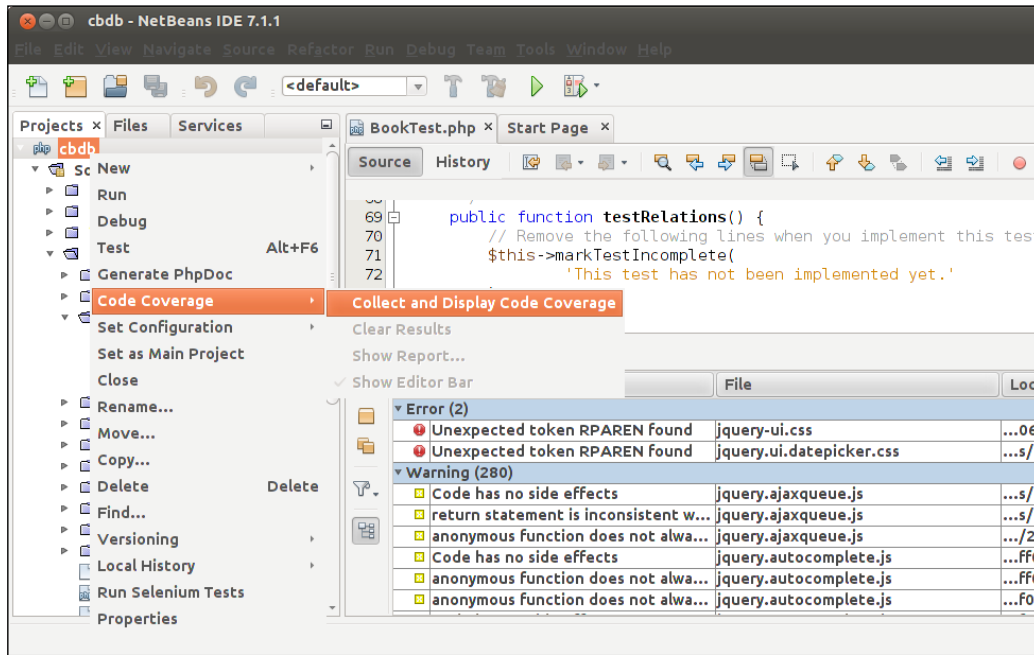
11. Run the tests again like we did in step 1, then get the code coverage report by right-clicking on the project root (**cbdb**) and selecting **Code Coverage | Show Report**.

Objective Complete - Mini Debriefing

There is so much more you can do with testing. We will revisit testing and introduce more tools and techniques in future chapters, but we encourage you to hone your testing and continue to explore the tools.

Classified Intel

When you generate the test coverage, PHPUnit marks the test stubs with "to-do" comments. To get the list of reminders of items you intended to complete, open the tasks list **Window | Tasks** (in other versions it may appear as **Window | Action Items**). It will provide you with a list of all errors and warnings in your code, as well as all to-do items.

Mission Accomplished

You have now successfully configured your development environment, generated and customized a Yii app, and begun to write your unit tests. This is a great baseline that you have created to work on the remaining projects in the book.

You Ready to go Gung HO? A Hotshot Challenge

Here are a few suggestions to go gung ho at this point:

- ▸ Customize the look of your site:
    - ❑ Change up the CSS.
    - ❑ Customize the landing page.

- ▸ Write a greeting for visitors.

- ▸ Write yourself a nice welcome for when you log in.

- ▸ Add validation to your forms.
    - ❑ What happens if you input a null value for **Publication Date**?
    - ❑ How is the security? Can you input illegal SQL in your text fields?

- ▸ Create your own user and deactivate the default accounts: `demo` and `admin`.

- ▸ Explore the tools that NetBeans provides. Could you have completed all of the steps of the project from within NetBeans? What hot keys do you find save the most time?

Project 2

Turn That DB into a Personal Mobile App

We will put a mobile face on our web application in this mission. Once your site is deployed to a production server, you might want to connect to it with your smartphone from a comic book store to check or update your collection.

Mission Briefing

The mission, should you choose to accept it, is to detect if a mobile browser is viewing the site, and if so, to display a new custom mobile view that we will create. We will need to configure a mobile device to connect to our local network, and then allow it to access our development website. We will go ahead and create UI controls that use the associations between book and author, book and illustrator, and book and publisher. We will add an "issue number" field to the book object, making our site more useful for comic book collectors.

Why Is It Awesome?

While regular websites are often usable with smartphones and tablets, mobile views make it much easier to use and navigate with these devices. Mobile views give your site a native-mobile look and feel. If you have ever tried to access something from your mobile device while you are in a hurry, you can appreciate the utility of good mobile interfaces.

Your Hotshot Objectives

▶ Setting Up Your Mobile Device

▶ Detecting Mobile Browser

▶ Creating a Mobile View

▶ Finishing Touches for the Mobile View

▶ Detecting Mobile Browser – The Real Deal

▶ Adding Issue Number to the Book Object

▶ Relationship Therapy

▶ Creating a Mobile View Widget

Mission Checklist

In order to follow this chapter, you need a network that you can connect to from both your development machine and a mobile device. If you don't have a device, you can use one of the various mobile development emulators. Check out the Google Android Developer SDK or Apple's iOS Dev Center if you want to get started on mobile development.

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, follow these steps, replacing the username jhamilton with your own username.

1. Copy the project files into your working directory.

   ```
   cp –r ~/Downloads/project_files/Chapter\ 2/project_files
       ~/projects/cbdb/ch2
   cd ~/projects/ch2/
   sudo chown -R jhamilton:www-data protected/runtime assets
   ```

2. If you already have a cbdb link in your webroot, delete it, so we're only looking at one project at a time (for now).

   ```
   sudo rm /opt/lampp/htdocs/cbdb
   ```

3. Create a link in the webroot directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   sudo ln -s /home/jhamilton/projects/ch2 cbdb
   ```

4. Import the project into NetBeans and configure for Yii development with PHPUnit.

5. Create a database named `cbdb` and load the database schema (`~/projects/cbdb/protected/data/schema.sql`) into it. If you already have a `cbdb` database from the previous chapter, you might want to back it up. In order to create a new `cbdb` database, the corresponding database with the same name from the first chapter has to be dropped.

6. If your web location is different, or if your access to MySQL is restricted, you will need to update the Yii configuration file (`~/projects/cbdb/ch2/protected/config/main.php`).

7. Download JQuery Mobile from `http://code.jquery.com/mobile/1.1.0/` (current stable release at the time of writing) and save it in `~/projects/cbdb/ch2/js/`.

```
cd projects/ch2/js
wget http://code.jquery.com/mobile/1.1.0/jquery.mobile-
  1.1.0.zip
unzip jquery.mobile-1.1.0.zip
cp jquery.mobile-1.1.0/ jquery.mobile-1.1.0.min.js .
cp jquery.mobile-1.1.0/jquery.mobile-1.1.0.min.css ../css/
cp -R jquery.mobile-1.1.0/images/ ../css/
rm jquery.mobile-1.1.0.zip
```

Setting Up Your Mobile Device

In this task, we set up your network so you can connect to your site with your mobile device. This will come in handy when we want to test the look and feel of the mobile version of the site.

Prepare for Lift Off

In order to test the parts of the site intended for mobile browsers, we will need to connect the computer hosting the CBDB app to a trusted Wi-Fi network, and then connect your mobile device to this network as well. The network should be encrypted and secured properly with a password, and should only allow connections from trusted entities.

Engage Thrusters

1. Follow the Wi-Fi connection instructions for your mobile device to connect it to your wireless network.

2. Make sure the computer with your LAMP stack is connected to the network as well.

3. You can determine the local IP address of your computer in Linux by typing `ifconfig` in the terminal.

43

4. Use your mobile device to look at the computer by putting the IP address in the mobile browser `http://192.168.3.23/cbdb`. You should see the Comic Book Database web application. If you cannot see the website, you may need to configure your router to allow network nodes to see each other.

Objective Complete - Mini Debriefing

Now we have set up a way to view and test the site from a mobile device.

Classified Intel

It is important to understand that we have set up a development environment, and not a production environment. There are many steps that are beyond the scope of this book that you should take to secure a production web server. For example, if you have gone with the default configurations so far, your installation of MySQL has no root password. I cannot state strongly enough that you should not use this configuration on the Internet or even an untrusted local network such as a coffee shop (or possibly your workplace). If these restrictions are unacceptable to you, you should at least configure XAMPP to only allow connections from localhost (you will also need to set it up to allow your mobile device to connect for the current task).

Detecting Mobile Browser

When a browser connects, we should determine if it is a mobile browser. We will need to write some code to do this.

Engage Thrusters

1. Open **ch2** | **SourceFiles** | **protected** | **views** | **layouts** | **main.php** and add these lines in the head section after the `css` includes and before the `title` tag:

```php
<?php
    Yii::app()->clientScript-
      >registerCoreScript('jquery');
    Yii::app()->clientScript->registerScriptFile(
      Yii::app()->request->baseUrl .
        '/js/detectmobilebrowser.js'
    );
  ?>
```

This will load jQuery (it comes with Yii) and our custom browser detection script.

2. Now, to test our handiwork, temporarily add the following lines at the beginning of the `mainmenu` div, after the opening `php` tag:

```
Yii::app()->clientScript-
  >registerScript('detectmobilebrowser',"
  if (isMobileBrowser(navigator.userAgent
    ||navigator.vendor||window.opera)) {
    alert('Mobile');
  }
  else {
    alert('Non-Mobile');
  }
  ",CClientScript::POS_READY);
```

3. Go ahead and open the site on your computer with your favorite browser. You should see the following screenshot:
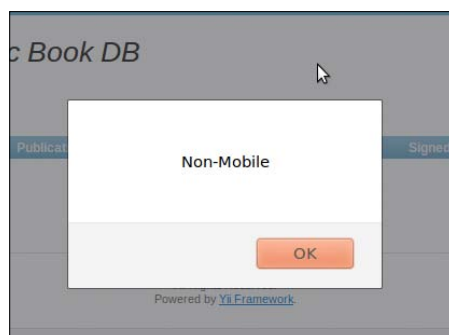
4. Now open the site in a browser on your mobile device. You should see the following screenshot:
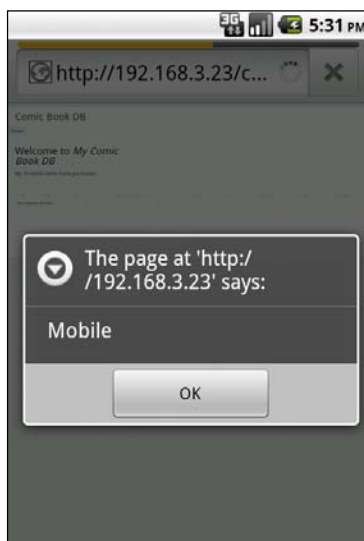
Objective Complete - Mini Debriefing

The `registerCoreScript` statement includes the jQuery package, so we can use jQuery in conjunction with JavaScript. The `registerScriptFile` statement makes code in `detectmobilebrowser.js` available to the layout. The `registerScript` statement runs the quoted snippet of JavaScript when the page is loaded. The JavaScript snippet checks to see if the requesting browser is a known mobile user-agent. If it is, an alert saying `Mobile` will be displayed. If the browser is not a known mobile user-agent, the alert displayed will say `Non-Mobile`. Any page that uses the main layout will display this. This change is just temporary, to check that we are properly detecting mobile browsers with JavaScript.

Classified Intel

There is a Yii extension named `detectmobilebrowser`. We could use this to detect if you are viewing the website with a cellular phone or another mobile device. At the time of writing, the extension can be found at `http://www.yiiframework.com/extension/detectmobilebrowser/`. However, there is a school of thought that believes mobile-browser detection belongs in client-side JavaScript rather than the controller. We will adhere to this school of thought and therefore will use JavaScript to detect this instead. We have borrowed detection code for jQuery from `http://detectmobilebrowsers.com`. We have included a modified version of this file, along with a copy of its very unrestrictive license, in `~/projects/cbdb/ch2/js/detectmobilebrowser.js`.

Creating a Mobile View

In this section, we will make a mobile view. We can test our handiwork by using the URL `mobile` to turn the mobile version of the layout on and off. In addition to this, we are going to add attributes to elements in our layouts to make them function properly with jQuery Mobile and make our mobile view look even better. jQuery Mobile uses particular attributes for HTML elements to determine the role and appearance of those elements in the jQuery Mobile page layout. There is a wonderful documentation about this, titled *Anatomy of a Page*, on `http://jquerymobile.com` in the documentation section about pages and dialogs.

Engage Thrusters

1. There is an empty skeleton of a layout at **ch2** | **Source Files** | **protected** | **views** | **layouts** | **mobile.php**. Open it.

2. After the following code snippet:

    ```
    <!--CSS includes here -->
    ```

3. Add the following line:

```
<link rel="stylesheet" type="text/css" href="<?php echo
   Yii::app()->request->baseUrl; ?>/css/jquery.mobile-
      1.1.0.min.css" media="screen, projection" />
```

4. After the following code snippet:

```
Yii::app()->clientScript->registerCoreScript('jquery');
```

Add the following line:

```
      Yii::app()->clientScript->registerScriptFile(
         Yii::app()->request->baseUrl . '/js/jquery.mobile-
            1.1.0.min.js');
```

5. Now, in the body section, add:

```
<div data-role="page">
  <div data-role="content">
  <?php echo $content; ?>
  </div><!-- content -->
  <div data-role="footer">
    <center>
      <?php echo Yii::powered(); ?>
    </center>
  </div><!-- footer -->
</div><!-- page -->
```

6. We are now going to override the base class `CController` to use the new mobile layout if the parameter `mobile` has a value of `on`. In order to do this, we will first replace the test code you wrote in the previous task in **ch2 | Source Files | protected | views | layouts | main.php** with code to redirect to the current URL, with the `mobile` parameter set to a value of `on`. Replace the following code:

```
if(isMobileBrowser(navigator.userAgent||navigator.vendor|
  |window.opera)) {
  alert('Mobile');
}
else {
  alert('Non-Mobile');
}
```

with:

```
if(isMobileBrowser(navigator.userAgent||navigator.vendor|
  |window.opera)) {
```

```
if (window.location.search.search('mobile') == -1) {
  if (window.location.search.length) {
    window.location.replace
      (document.URL + '&mobile=on');
  }
  else {
    window.location.replace
      (document.URL + '?mobile=on');
  }
}
}
```

7. Add the following function to **ch2 | Source Files | protected | components | Controller.php**:

```
/* Override beforeAction() to change to the mobile
   layout if URL param['mobile'] == 'on' */
protected function beforeAction($action) {
  if (Yii::app()->getRequest()->getQuery('mobile')
    == 'on') {
    Yii::app()->user->setState('mobile', true);
  }
  else if (Yii::app()->getRequest()->getQuery('mobile')
    == 'off') {
    Yii::app()->user->setState('mobile', false);
  }
  if (Yii::app()->user->getState('mobile')) {
    $this->layout = '//layouts/mobile';
  }
  return true;
}
```

8. If all has gone well, you should now be able to switch between `http://localhost/cbdb/` and `http://localhost/cbdb/?mobile=on` and see different layouts (see the following screenshots).

Full layout:

Mobile layout:

9. Open **ch2** | **Source Files** | **protected** | **views** | **layouts** | **mobile.php** and add the following lines between `<div data-role="page">` and `<div data-role="content">`:

```php
<?php
  $htmlOptions = array('data-role' => 'controlgroup',
    'class' => 'localnav');
  $linkOptions = array('data-role' => 'button',
    'data-theme' => 'b', 'rel' => 'external');
  $items = array();
  if (Yii::app()->user->isGuest) {
    $items[] = array('label'=>'Login',
      'url'=>array('/site/login'), 'linkOptions' =>
      $linkOptions);
  }
  else {
    $items[] = array('label'=>'Home',
      'url'=>array('/site/index'), 'linkOptions' =>
      $linkOptions);
    $items[] = array('label'=>'Comic Books',
      'url'=>array('/book'), 'linkOptions'=>
      $linkOptions);
    $items[] = array('label'=>'Logout (' . Yii::app()-
      >user->name . ')', 'url'=>array('/site/logout'),
      'linkOptions' => $linkOptions);
  }
  $non_mobile_uri = preg_replace('/mobile=on/',
    'mobile=off', /*'/site/login');*/Yii::app()->request-
    >baseUrl);
  $items[] = array('label'=>'Turn off mobile view',
    'url'=>array('?mobile=off'), 'linkOptions' =>
    $linkOptions);
  $this->widget('zii.widgets.CMenu',array(
    'activeCssClass' => 'active',
    'activateParents' => true,
    'htmlOptions' => $htmlOptions,
    'items'=> $items
  )
);
?>
```

10. If you switch to mobile view and then click on the link to log in, you will notice that the form is broken when you try to submit. This is because jQuery mobile needs one additional attribute to tell it how this form is intended to be used. Open the file **ch2 | Source Files | protected | views | site | login.php**, and, in your declaration of the CActiveForm widget, below the line `'enableClientValidation'=>true`, add the following line of code:

```
'htmlOptions' => array('data-ajax' => 'false'),
```

11. The whole declaration should now look like the following code snippet:

```
$form=$this->beginWidget('CActiveForm', array(
  'id'=>'login-form',
  'enableClientValidation'=>true,
  'htmlOptions' => array('data-ajax' => 'false'),
  'clientOptions'=>array(
    'validateOnSubmit'=>true,
  ),
));
```

12. Now, the mobile version of the login form should work. The non-mobile version of this form should also continue working the way it always has.

Objective Complete - Mini Debriefing

We included the CSS for jQuery Mobile and registered the JavaScript library so we could use it in our layout. We updated the main layout to notice if a mobile browser is detected and if the URL parameter mobile isn't set. If so, then we set the URL parameter `mobile` to `on` and redirect it back to the current URL.

We put a `beforeAction` action in components/controller (the subclass of CController that is provided to customize and override CController behavior) to check to see if the mobile parameter is set to `on`, and if so, to set the state of `mobile` to `true` in the session. Putting the mobile browser detection in the view and setting the mobile parameter abstracts the controller from mobile detection. By doing it this way, we break the problem into smaller pieces, and gain the ability to also manually select which layout we are using. At a later time, we can add a link that is labeled **Click for non-mobile view** to the mobile layout.

51

We've made changes to the instantiation of the CMenu widget in our mobile view to add attributes that jQuery Mobile uses for rendering. The data-role `localnav` lets jQuery know that this menu is for navigating our site. For the buttons, setting data-theme to `b` tells jQuery Mobile to use the built-in blue theme for the buttons. If you don't set `rel` to `external`, the links won't work properly because jQuery mobile will expect the link-targets to contain jQuery Mobile specific divs and data-roles that are not there. We've made a small change to our login form to let jQuery Mobile know how to submit the form. If you click on **Comic Books** in the mobile view, you'll notice that we've lost the operations menu for the books, and the list needs some sprucing up. We're going to fix that in the next task.

Classified Intel

The JavaScript we placed in the main layout to detect a mobile device and set the `mobile` parameter is by no means fool proof or robust. For example, the value of another parameter in the query string could be `mobile`, and the code wouldn't set `mobile` to `on` because of this. We have simply put it there as it is to serve as a proof of concept. Ultimately, we will rework this piece of code with JavaScript functions for processing URL parameters.

Finishing Touches for the Mobile View

With a few changes to the layout, we can get to where we need to go.

Engage Thrusters

1. Open **ch2** | **Source Files** | **protected** | **views** | **layouts** | **mobile.php**. Change the lines between `<div data-role="page">` and `<div data-role="content">` as follows:

```
<div data-role="collapsible" data-theme="b">
  <h3>Main Menu</h3>
  <?php
  $htmlOptions = array('data-role' => 'controlgroup',
    'class' => 'localnav');
  $linkOptions = array('data-role' => 'button',
    'data-theme' => 'b', 'rel' => 'external');
  $items = array();
  if (Yii::app()->user->isGuest) {
    $items[] = array('label'=>'Login',
      'url'=>array('/site/login'), 'linkOptions' =>
      $linkOptions);
  }
  else {
    $items[] = array('label'=>'Home',
      'url'=>array('/site/index'), 'linkOptions' =>
      $linkOptions);
```

```php
    $items[] = array('label'=>'Comic Books',
      'url'=>array('/book'), 'linkOptions'=>
      $linkOptions);
    $items[] = array('label'=>'Logout (' . Yii::app()-
      >user->name . ')', 'url'=>array('/site/logout'),
      'linkOptions' => $linkOptions);
  }
  $non_mobile_uri = preg_replace('/mobile=on/',
    'mobile=off', /*'/site/login');*/Yii::app()->request-
    >baseUrl);
  $items[] = array('label'=>'Turn off mobile view',
    'url'=>array('?mobile=off'), 'linkOptions' =>
    $linkOptions);
  $this->widget('zii.widgets.CMenu',array(
      'activeCssClass' => 'active',
      'activateParents' => true,
      'htmlOptions' => $htmlOptions,
      'items'=> $items
    )
  );
  ?>
</div><!-- collapsible -->
<?php
if (count($this->menu) > 0) {
  echo "<div data-role='collapsible' data-theme='b'>\n";
  echo "\t<h3>Operations</h3>\n";
  foreach ($this->menu as $key=>$item) {
    $this->menu[$key]['linkOptions'] = $linkOptions;
  }
  $this->widget('zii.widgets.CMenu', array(
    'items'=>$this->menu,
    'htmlOptions'=> $htmlOptions,
  ));
  $this->endWidget();
  echo "</div><!-- collapsible -->\n";
}?>
```

2. Reload the **Comic Books** page while logged in, with the mobile view on.

Objective Complete - Mini Debriefing

We made the decision to use collapsible menus to alleviate the cumbersome task of dealing with a large number of menu items. To do this, we create a div with a data-role of `collapsible` for each collapsible menu. We use the `<h3>` tag to indicate the title of the top-level collapsible control. If an object has operations that can be performed on it, it adds the items to `$this->menu`, so we check that to see if it has elements in the second PHP segment. If it does, we build and display the items.

Detecting Mobile Browser – The Real Deal

In the previous tasks, we have quickly added mobile browser detection. In this task, we will refine that code.

Prepare for Lift Off

The file **ch2 | Source Files | js | url_param_proc.js** contains functions for manipulating the parameter strings in the URL query string. We will include this file in our mobile layout and use it to manage the `mobile` parameter.

Open **ch2 | Source Files | protected | views | layouts | main.php**. After the following code snippet:

```
Yii::app()->clientScript->registerScriptFile(
Yii::app()->request->baseUrl . '/js/detectmobilebrowser.js'
);
```

Add the following lines of code:

```
Yii::app()->clientScript->registerScriptFile(
Yii::app()->request->baseUrl . '/js/url_param_proc.js'
);
```

Engage Thrusters

1. The file **ch2 | Source Files | js | url_param_proc.js** contains the following block of code:

```
function get_param_array() {
  var param_array = {};
  if (window.location.search.length) {
    var query_string =
      window.location.search.substring(1);
    var params = query_string.split("&");
    for (var count = 0; count < params.length; count++) {
      var param_pair = params[count].split("=");
      param_array[param_pair[0]] = param_pair[1];
    }
  }
  return param_array;
}
```

```
function build_query_string(param_array) {
  var query_string = "";

  for (key in param_array) {
    query_string += key + "=" + param_array[key];
  }
  if (query_string.length) {
    query_string = "?" + query_string;
  }
  return query_string;
}


function get_base_uri() {
  var base_uri = document.location.protocol + "//" +
    document.location.hostname;
  if (document.location.port.length) {
    base_uri += ":" + document.location.port;
  }
  base_uri += document.location.pathname;
  return base_uri;
}
```

The `get_param_array()` function processes the query string and returns an object with the keys as the parameter names and the values as the parameter values. The `build_query_string()` function reassembles the object back into a query string. The `get_base_uri()` function returns the URL without the query portion of the string.

2. In **ch2 | Source Files | protected | views | layouts | main.php**, change the following code snippet:

```
if
(isMobileBrowser(navigator.userAgent||navigator.vendor
  ||window.opera)) {
  if (window.location.search.search('mobile') == -1) {
    if (window.location.search.length) {
      window.location.replace
        (document.URL + '&mobile=on');
    }
    else {
      window.location.replace
        (document.URL + '?mobile=on');
    }
  }
}
```

To look like this:

```
if (isMobileBrowser(navigator.userAgent||navigator.vendor
  ||window.opera)) {
  var param_array = get_param_array();
  if (!('mobile' in param_array)) {
    param_array['mobile'] = 'on';
    window.location.replace(get_base_uri() +
      build_query_string(param_array));
  }
}
```

3. Test this by opening the site in your mobile browser.

Objective Complete - Mini Debriefing

We used some small Javascript functions to help us manage the URL parameters. It has helped us clean up our code and the code is now more resistant to unexpected results. The code basically does the same thing we intended for it to do before, but now our check for the presence of the `mobile` parameter differentiates between keys and values, and we build the query string in a more elegant fashion.

Adding Issue Number to the Book Object

We will add a field for `issue number` to our book object by adding it as a column for the already existing book table in our `cbdb` database. Since we have already created our model, view, and controller with Gii, we will need to manually modify these files to use the new field.

Engage Thrusters

1. In the **Services** tab, open the connection to **cbdb** and open the list of tables:
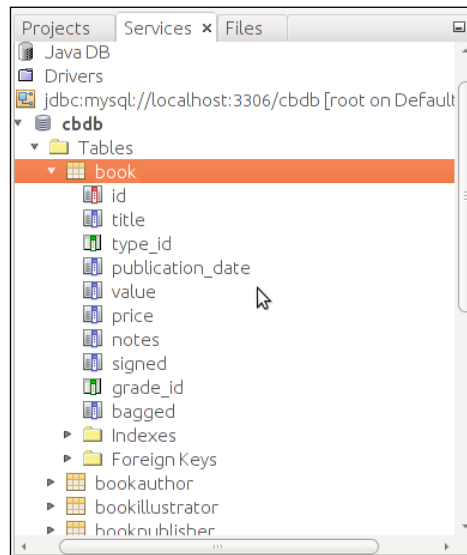
2. Right-click on the **book** table and go to **Add Column**. Name the field `issue_number`, select **VARCHAR** for the type and `10` for the size, and click on **OK** as shown in the following screenshot:
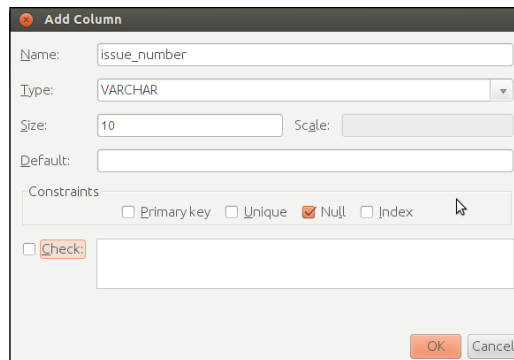
3. Now let's make the model aware of our change. Open **ch2** | **Source Files** | **protected** | **models** | **Book.php**. In the comment block at the top of the file, add the field description below `* @property integer $bagged`:

   ```
   * @property integer $issue_number
   ```

4. In the `rules()` function, add the issue number in the relevant places:

   ```
   return array(
     array('title', 'required'),
     array('signed, bagged', 'numerical',
       'integerOnly'=>true),
     array('title', 'length', 'max'=>256),
     array('type_id, value, price, grade_id', 'length',
       'max'=>10),
     array('publication_date, notes', 'safe'),
     array('issue_number', 'length', 'max'=>10),
     // The following rule is used by search().
     // Please remove those attributes that should not be
     //searched.
     array('id, title, type_id, publication_date, value,
       price, notes, signed, grade_id, bagged,
       issue_number', 'safe', 'on'=>'search'),
   )
   ```

5. In the `attributeLabels()` function, once again add the appropriate information for issue_number:

   ```
   return array(
     'id' => 'ID',
     'title' => 'Title',
     'type_id' => 'Type',
     'publication_date' => 'Publication Date',
     'value' => 'Value',
     'price' => 'Price',
     'notes' => 'Notes',
     'signed' => 'Signed',
     'grade_id' => 'Grade',
     'bagged' => 'Bagged',
     'issue_number' => 'Issue Number',
   );
   ```

6. Finally, in the `search()` function, add the criterion for the field:

   ```
   $criteria->compare('signed',$this->signed);
   $criteria->compare('grade_id',$this->grade_id,true);
   $criteria->compare('bagged',$this->bagged);
   ```

```
$criteria->compare('issue_number', $this->issue_number,
   true);

return new CActiveDataProvider($this, array(
   'criteria'=>$criteria,
));
```

7. Now that the model knows about `issue_number`, we need to add it to the book views. Open **ch2** | **Source Files** | **protected** | **views** | **book** | **_view.php** and add the following highlighted lines:

```
<b><?php echo CHtml::encode
   ($data->getAttributeLabel('title')); ?>:</b>
<?php echo CHtml::encode($data->title); ?>
<br />

<b><?php echo CHtml::encode
   ($data->getAttributeLabel('issue_number')); ?>:</b>
<?php echo CHtml::encode($data->issue_number); ?>
<br />

<b><?php echo CHtml::encode
   ($data->getAttributeLabel('type_id')); ?>:</b>
<?php echo CHtml::encode($data->type_id); ?>
<br />
```

8. Now open **ch2** | **Source Files** | **protected** | **views** | **book** | **_form.php**. This view shows what fields will be displayed for create and update. Add the following code snippet:

```
<div class="row">
   <?php echo $form->labelEx($model,'title'); ?>
   <?php echo $form->textField($model,'title',array
      ('size'=>60,'maxlength'=>256)); ?>
   <?php echo $form->error($model,'title'); ?>
</div>

<div class="row">
   <?php echo $form->labelEx($model,'issue_number'); ?>
   <?php echo $form->textField($model,'issue_number',array
      ('size'=>20,'maxlength'=>10)); ?>
   <?php echo $form->error($model,'issue_number'); ?>
</div>

<div class="row">
```

```php
    <?php echo $form->labelEx($model,'type_id'); ?>
    <?php echo $form->dropDownList
      ($model, 'type_id', $model->getTypeOptions()); ?>
    <?php echo $form->error($model,'type_id'); ?>
</div>
```

9. Now do the same for the search by opening **ch2 | Source Files | protected | views | book | _search.php** and adding the following code snippet:

```php
<div class="row">
  <?php echo $form->label($model,'title'); ?>
  <?php echo $form->textField($model,'title',array
    ('size'=>60,'maxlength'=>256)); ?>
</div>

<div class="row">
  <?php echo $form->label($model,'issue_number'); ?>
  <?php echo $form->textField($model,'issue_number',array
    ('size'=>20,'maxlength'=>10)); ?>
</div>

<div class="row">
  <?php echo $form->label($model,'type_id'); ?>
  <?php echo $form->textField($model,'type_id',array
    ('size'=>10,'maxlength'=>10)); ?>
</div>
```

10. Add the field after **title in ch2 | Source Files | protected | views | book | view.php**:

```php
<?php $this->widget('zii.widgets.CDetailView', array(
  'data'=>$model,
  'attributes'=>array(
    'id',
    'title',
    'issue_number',
    'type_id',
    'publication_date',
    'value',
    'price',
    'notes',
    'signed',
    'grade_id',
    'bagged',
  ),
)); ?>
```

11. Add the field after title in **ch2** | **Source Files** | **protected** | **views** | **book** | **admin. php** and in **ch2** | **Source Files** | **protected** | **views** | **site** | **index.php**.

12. Let's see where this puts us.

13. The following is the create/update form:

14. The following is the mobile version of the create/update form, also with issue number added and working:

15. The following is the index page:
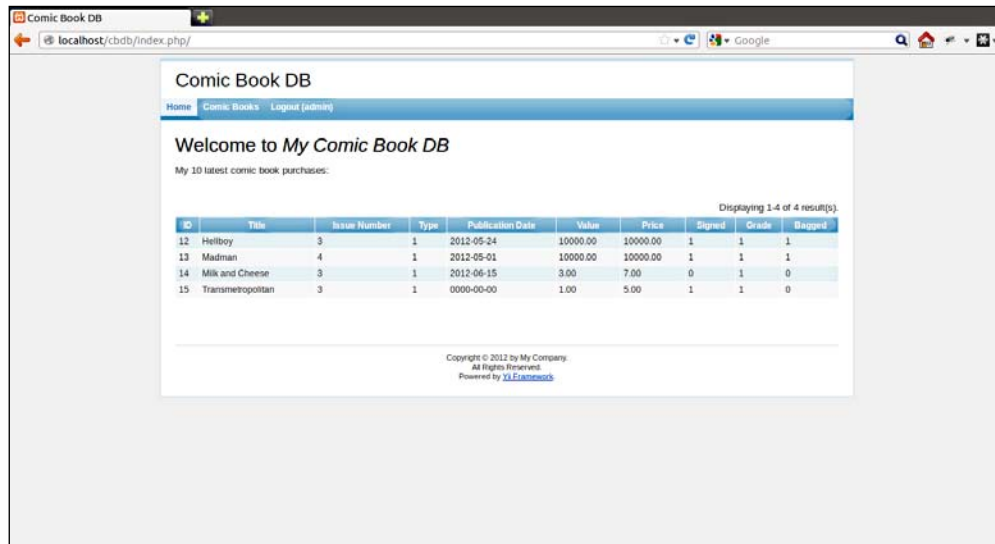
16. The following screenshot is the mobile view of the same page:
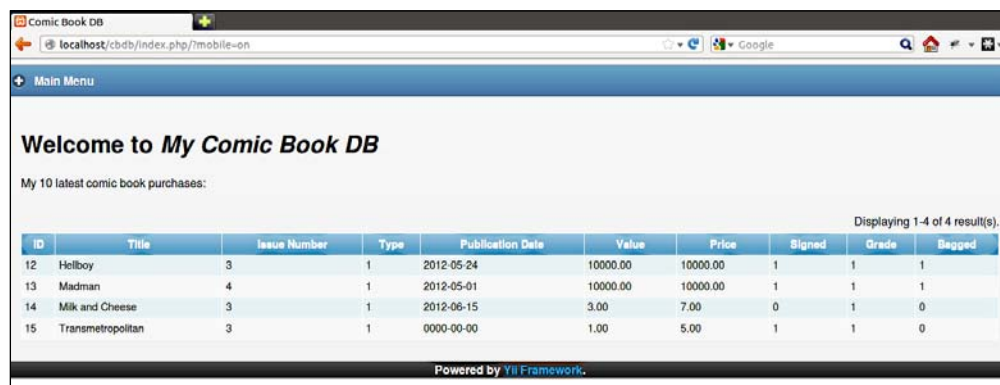
We have successfully added a column manually for issue number. It shows up on our forms, our views, and it flows from the UI to the database, and back. We were able to accomplish all of this with relatively little effort, due to the power of Yii's well-implemented MVC design pattern (`http://www.yiiframework.com/doc/guide/1.1/en/basics.mvc`).

Objective Complete - Mini Debriefing

In the model, `rules()` provides low-level rules for field-level validation. We limit the length of `issue_number` to 10, because we set its length to 10 in the database. The `attributeLabels()` function defines the labels for each field. In `search()`, we added a criterion for issue number. In the Yii documentation, the prototype for `CDbCriteria.compare` is as follows:

```
public CDbCriteria compare(string $column, mixed $value, boolean
    $partialMatch=false, string $operator='AND',
    boolean $escape=true)
```

These are the only changes we needed to make for the model to correctly use and expose `issue_number`. We then changed the views, in a very obvious fashion, to allow them to use the same field. The view `_form.php` is shared by `create.php` and `update.php`, via `$this->renderPartial()`. In the file `index.php`, `_view.php` is used. We manually added `issue_number` to the field lists in the other `views`.

Relationship Therapy

We will make use of the books' many-to-many relationship with authors. We will have to make moderately extensive changes to the model, the controller, and pertinent views. Hang on to your hat!

Engage Thrusters

1. Update the book model to capture the author relationship (open **ch2 | Source Files | protected | models | Book.php**):

```
'authors' => array(self::MANY_MANY, 'Person',
    'bookauthor(author_id, book_id)', 'index'=>'id'),
    'bookauthors' => array(self::HAS_MANY, 'BookAuthor',
    'book_id', 'index' => 'author_id'),
```

2. Update `loadModel()` in the book controller to include related author data (open **ch2 | Source Files | protected | controllers | BookController.php**):

```
$model=Book::model()->with("authors")->findByPk($id);
```

3. Add an authors display to the book edit form in **ch2 | Source Files | protected | views | book | _form.php**.

```php
<div class="row">
  <?php echo $form->labelEx($model,'author'); ?>
  <?php
    echo "<ul class=\"authors\">";
    foreach ($model->authors as $auth) {
      echo "<li>" . CHtml::encode($auth->fname . " " .
        $auth->lname) . "</li>";
    }
    echo "</ul>";
  ?>
</div>
```

4. This will display the author(s) associated with a comic book. You can view the results by clicking on **book number 4**. It has an author already associated with it. Next, we will update the form so we can add authors.

5. Make an `addAuthor()` function in the book model.

```php
public function addAuthor($author) {
  if ($author->isNewRecord()) {
    $author->save();
    $bookauthor = new BookAuthor();
    $bookauthor->book_id = $this->id;
    $bookauthor->author_id = $author->id;
    $bookauthor->save();
  }
}
```

6. Add a `createAuthor()` function to the book controller.

```php
protected function createAuthor($book) {
  $author = new Person();

  if(isset($_POST['Person'])) {
    $author->attributes=$_POST['Person'];
    if ($book->addAuthor($author)) {
      Yii::app()->user->setFlash('authorAdded',
        "Added author " . CHtml::encode($author->fname .
        " " . $author->lname));
      $this->refresh();
    }
  }
  return $author;
}
```

7. Include the call to `createAuthor()` in the book controller `actionUpdate()`.

```
public function actionUpdate($id)
{
    $model=$this->loadModel($id);
    $author= $this->createAuthor($model);
```

8. Add the result to the call to render at the end of the action.

```
    $this->render('update',array(
        'model'=>$model,
        'author'=>$author,
    ));
```

9. Similarly, update the create action in the book controller to include.
   `createAuthor()`:

```
public function actionCreate()
{
    $model=new Book;
    $author= $this->createAuthor($model);
```

10. Pass the author value to render.

```
    $this->render('create',array(
        'model'=>$model,
        'author'=>$author,
    ));
```

11. Add a field for the new author name to the book form.

```
<?php echo $form->labelEx($model,'author'); ?>
<?php if(Yii::app()->user->hasFlash('authorAdded')) { ?>
    <div class="flash-success">
        <?php echo Yii::app()->user->getFlash
            ('authorAdded'); ?>
    </div>
<?php } else {
    echo $this->renderPartial('/person/_form', array(
        'model' => $author,
    ));
        } ?>
<?php
    echo "<ul class=\"authors\">";
    foreach ($model->authors as $auth) {
        echo "<li>" . CHtml::encode($auth->fname . " " .
            $auth->lname) . "</li>";
    }
    echo "</ul>";
?>
```

12. Update the last line in both `create.php` and `update.php` to pass author object to `renderPartial()`.

```php
<?php echo $this->renderPartial('_form',
  array('model'=>$model, 'author' =>$author)); ?>
```

13. Create a partial file named `_li.php` for the author list element and add a delete button to each element.

```php
<?php
    echo "<li id=\"author-" . $author->id. "\">" .
        CHtml::encode($author->fname . " " .
        $author->lname) .
        " <input class=\"delete\" " .
            "type=\"button\" url=\"" .
    Yii::app()->controller->createUrl("removeAuthor",
            array("id" => $model->id,
            "author_id"=>$author->id,
            "ajax"=>1)) .
        "\" author_id=\"". $author->id.
        "\" value=\"delete\" />" .
        "</li>";
```

14. Update the book form to call `renderPartial()` to render the list element.

```php
<?php
    if (count($model->authors)) {
      echo "<ul class=\"authors\">";
      foreach ($model->authors as $auth) {
        echo $this->renderPartial('_li', array(
                'model' => $model,
                'author' => $auth,
          ));
      }
      echo "</ul>";
    }
?>
```

15. Add the `primaryKey()` function to the `BookAuthor` model.

```php
public function primaryKey()
{
  return array('book_id', 'author_id');
}
```

16. Add the `removeAuthor()` function to the book model.

```
public function removeAuthor($author_id) {
  $pk = array('book_id'=>$this->id, 'author_id' =>
    $author_id);
  BookAuthor::model()->deleteByPk($pk);
}
```

17. Add the `removeAuthor()` action to the book controller.

```
public function actionRemoveAuthor($id) {
  // request must be made via ajax
  if(Yii::app()->request->isAjaxRequest()) {
    $model=$this->loadModel($id);
    $model->removeAuthor($_GET['author_id']);
  }
  else {
    throw new CHttpException(400,'Invalid request.');
  }
}
```

18. Add the `removeAuthor` action to the list of actions requiring an authenticated user.

```
'actions'=>array('create','update', 'removeAuthor'),
```

19. Change the person form to make it work in the create form and to use AJAX to submit in the update form.

```
<div class="ajax-form">
    <div class="row">
    <?php echo CHtml::activeLabel($model,'fname'); ?>
    <?php echo CHtml::activeTextField
      ($model,'fname',array
      ('size'=>32,'maxlength'=>64)); ?>
    <?php echo CHtml::activeLabel($model,'lname'); ?>
    <?php echo CHtml::activeTextField
      ($model,'lname',array
      ('size'=>32,'maxlength'=>64)); ?>
  </div>
</div><!-- form -->
```

20. Record the book/author association in the book create action.

```
if($model->save()) {
  // record book/author association
  $ba = new BookAuthor;
  $ba->book_id = $model->id;
  $ba->author_id = $author->id;
  $ba->save();

  $this->redirect(array('view','id'=>$model->id));
}
```

21. Add a submit button to add authors on the update form.

```php
<?php } else {
    echo $this->renderPartial('/person/_form', array(
            'model' => $author,
            'subform' => 1
        ));
    if (Yii::app()->controller->action->id != 'create') {
?>
        <div class="row buttons">
        <input class="add" type="button"
            obj="Person"
            url="<?php
            echo Yii::app()->controller->createUrl(
                "createAuthor",
                array("id"=>$model->id)); ?>"
            value="Add"/>
        </div>
    <?php }

} ?>
```
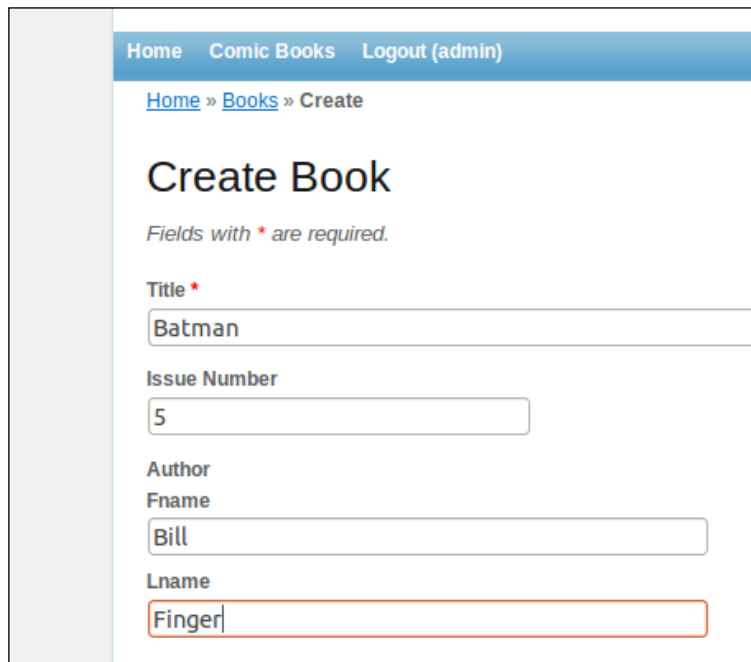
22. Add the `createAuthor` action to the book controller.

```php
public function actionCreateAuthor($id) {
  // request must be made via ajax
  if(isset($_GET['ajax']) && isset($_GET['Person'])) {
    $model=$this->loadModel($id);
    $author = new Person();
    $author->attributes=$_GET['Person'];
    if (($author->fname != null) &&
        ($author->lname !=null) )
    {
      $model->addAuthor($author);
      $this->renderPartial('_li',array(
          'model'=>$model,
          'author'=>$author,
      ), false, true);
    }
  }
  else {
    throw new CHttpException(400,'Invalid request.');
  }
}
```

23. Add `createAuthor` to authorized actions.

```
array('allow', // allow authenticated user to perform
  'create' and 'update' actions
  'actions'=>array('create','update', 'removeAuthor',
    'createAuthor'),
  'users'=>array('@'),
),
```

24. Let's see what we've got. The create form is as follows:

The revised update form is as follows:

Update Book 14

*Fields with * are required.*

Title *

Road Rage

Issue Number

1

Author
Fname

Joe

Lname

Hill

[Add]

- Stephen King [delete]

Type

trade

Objective Complete - Mini Debriefing

As you cannot add multiple authors to an object until you have created it, for now we put this capability on the update form. This task has shown us how to make extensive changes to the existing Yii infrastructure, and gives a feel for the kind of real work you can expect to do with the framework.

Creating a Mobile View Widget

We will create a widget to customize the list view for books. When we are finished, the mobile view for the book list will look a lot better.

Engage Thrusters

1. We will create a directory for our extension to keep everything together. Create an extensions directory in `~/ch2/protected` as follows:

    cd ~/projects/cbdb/ch2/protected

    mkdir extensions

2. Make a directory for the widget under extensions named `mobile`.

    cd ~/projects/cbdb/ch2/protected/extensions

    mkdir mobile

3. In the widget directory, create a file named `ListView` with an init and run function. The init function will prepare any assets that your view needs, but our mobile layout has already taken care of this for us. The run function will render the widget.

```php
<?php
class ListView extends CWidget
{
  public $dataProvider;
  public $itemView;

  public function init()
  {
    parent::init();
    // add any assets here
  }

  public function run()
  {
    parent::run();
    if($this->dataProvider===null)
      throw new CException(Yii::t('ext.mobile',
        '"dataProvider" field must be set.'));
      if($this->itemView===null)
        throw new CException(Yii::t('ext.mobile',
          '"itemView" field must be set.'));

    $this->render('body');
  }
}
```

4. Create a view directory.

    cd ~/ch2/protected/extensions/mobile

    mkdir views

5. Create the view for the widget in the views directory (**ch2** | **Source Files** | **protected** | **extensions** | **mobile** | **views** | **body.php**). The view will bracket our data in a jQuery mobile list and render our `itemView` template in each list element.

```
<ul data-role="listview" data-theme="g">
<?php
    $data = $this->dataProvider->getData();
    $owner = $this->getOwner();
    foreach ($data as $i=>$item) {
      echo "<li>";
      $owner->renderPartial($this->itemView, $item);
      echo "</li>";
    }
?>
</ul>
```

6. Copy the `_view.php` view file in the book view directory to a file named `_mview.php`. Edit `_mview.php`. Remove the div tags and the headers. Remove all fields except for title and notes. Add an h1 tag around title and a p tag around notes. Put the whole thing inside a single PHP tag so it looks like this:

```
<?php
  echo "<h1>" . CHtml::encode($data->title) . "</h1>";
  echo "<p>" . CHtml::encode($data->notes) . "</p>";
?>
```

7. Copy the `index.php` view file in the book view directory to a file named `mobile_index.php`. Remove the header, change the widget call from `zii.widgets.CListView` to our new widget, and change the item view to `_mview.php`. Put everything in a single set of PHP tags. The file will look like this:

```
<?php
$this->menu=array(
    array('label'=>'Create Book',
      'url'=>array('create')),
    array('label'=>'Manage Book', 'url'=>array('admin')),
);

$this->widget('ext.mobile.ListView', array(
    'dataProvider'=>$dataProvider,
    'itemView'=>'_mview',
)); ?>
```

8. Update the index action in the book controller to render the mobile view if access is from a mobile device.

```php
public function actionIndex()
{
  $view = 'index';
  $dataProvider=new
  CActiveDataProvider('Book');
  if (Yii::app()->user->getState('mobile')) {
    $view = 'mobile_index';
  }
  $this->render($view,array(
    'dataProvider'=>$dataProvider,
  ));
}
```

9. At this point, the book index should display a nice readable list of books.

10. To make the list more manageable, we can add a search feature. jQuery makes this easy by providing built-in support for a list search. Just add `data-filter="true"` to the list tag.

```html
<ul data-role="listview" data-theme="g"
    data-filter="true">
```

11. Let's look at what we have now:
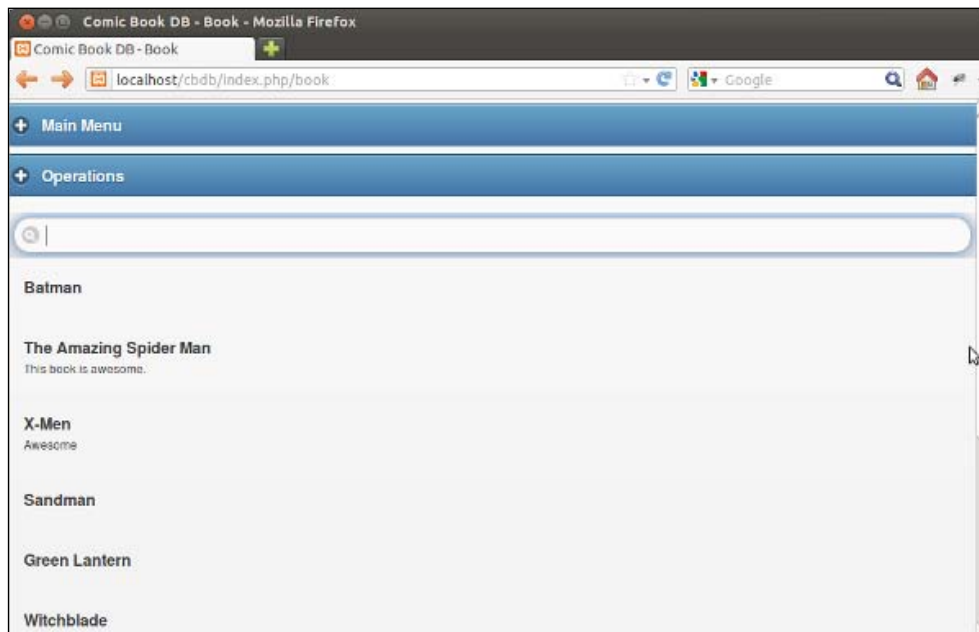
12. The list looks a *lot* nicer now. Let's see how our mobile-optimized, all-in-one search filter works.

13. It is awesome, indeed. Try a different filter:

That is some powerful stuff.

14. Let's add issue numbers to this new view. You could then use this view to see what books you have while you are shopping at the comic book store.

15. Open **ch2 | Source Files | protected | views | book | _mview.php** again and change it up:

```php
<?php
  echo '<h1>' . CHtml::encode($data->title);
  if (!is_null($data->issue_number)) {
    echo  ' ' . CHtml::encode($data->issue_number);
  }
  echo '</h1>';
  echo '<p>' . CHtml::encode($data->notes) . '</p>';
?>
```

16. Better yet, let's change the view so that you can click on the list items to pull up the detailed view of the record. Look at what your previous changes did, then try changing it to the following code snippet:

```php
<?php
  echo '<a href="/cbdb/index.php/book/' .
    $data->id . '">';
  echo '<h1>' . CHtml::encode($data->title);
  if (!is_null($data->issue_number)) {
    echo ' Issue: ' . CHtml::encode
      ($data->issue_number);
  }
  echo '</h1>';
  echo '<p>' . CHtml::encode($data->notes) . '</p></a>';
?>
```

17. After adding the preceding code, the list will look like the following screenshot:
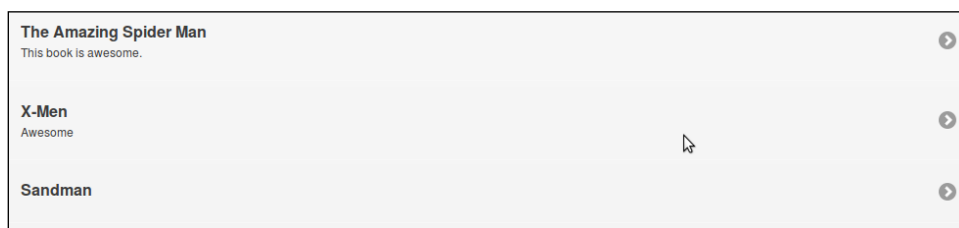
Here is what the detailed view looks like:

This is a useful set of changes.

Objective Complete - Mini Debriefing

Let's look at what we've accomplished: We created a widget for listing book objects in the mobile view and called it `ListView`, we made changes necessary to provide full mobile functionality to the list view for comic books, and we added a slick mobile search.

Mission Accomplished

We have learned a great deal about adding mobile functionality to a Yii project. We have seen how to include jQuery Mobile and use it in our layouts, views, and forms. We know how to make a nice mobile search. We have examples of how to add functionality and fields to an existing Yii project.
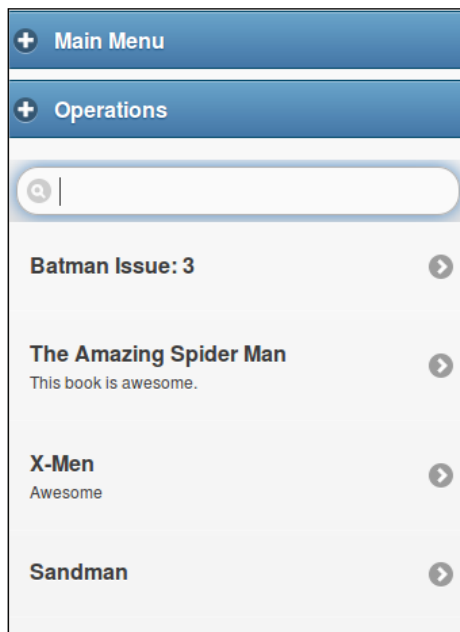
You Ready to go Gung HO? A Hotshot Challenge

Here are some suggestions to try for yourself with this project:

- ▶ Mobile optimization

  - ❑ Try out your mobile view with several different mobile devices. Does it work for tablets? If you find unsupported devices, extend the device identifier algorithm.

  - ❑ Review the forms and other pages of the app and optimize them for mobile viewing.

- ▶ Book form extending

  - ❑ Add confirmation dialog to author delete action

  - ❑ Add Ajax error handling to author add and delete actions

  - ❑ Add Ajax confirmation flash for author add and delete actions

  - ❑ Include publisher, illustrator, and tag fields in book form

- ▶ Book view perfecting

  - ❑ Update Book view to remove extraneous information like ID and include related information like author, illustrator, and publisher

- ▶ Mobile App Expanding

  - ❑ Expand the book list to allow you to group by publisher, illustrator, tag, and so on

Project 3

Access All Areas – Users and Logins

In this mission, we will replace the default Yii user management and access control with a database-driven implementation, and then we will apply the access control to a new site function, and verify our work with some tests.

Mission Briefing

We will add a user table to the application database, and then generate the Yii scaffolding and customize it. We will extend the user management interface to utilize our user table fields. We will add a new feature to the site – a wish list viewer for friends and family looking for gift ideas, and then create friends and family users and give them access to the wish list. When we are done, we will be able to assign different capabilities to different users, and their menus will reflect the actions they are permitted to take. For example, guest users will only be able to read comic book entries, not add, edit, or delete, as the menu in the following screenshot demonstrates:
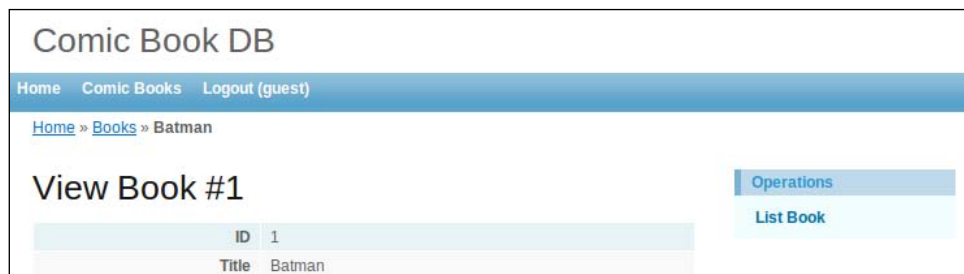
Why Is It Awesome?

The generated Yii project files include a basic access control system to help you start building your project. However, if your project requirements include providing access to a large number of users, you will soon find it helpful to include user management in your site. There are some great Yii extensions available that provide user management. These may be more or less what you want. If your project needs are unique or you would just like to take a tour through a user management implementation, this chapter will be of interest.

Your Hotshot Objectives

- ▶ Adding a User Object with CRUD
- ▶ Making a User Management Interface
- ▶ Storing Passwords
- ▶ Activating Database User Login
- ▶ Enforcing Secure Passwords
- ▶ Adding User Functions – Wishlist
- ▶ Configuring User Access
- ▶ User Specific Menus

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, carry out the following steps replacing the username `lomeara` with your own username:

1. Copy the project files into your working directory.

   ```
   cp –r ~/Downloads/project_files/Chapter\ 3/project_files
     ~/projects/ch3
   ```

2. Make the directories that Yii uses web writeable.

   ```
   cd ~/projects/ch3/
   sudo chown -R lomeara:www-data protected/runtime assets
     protected/models protected/controllers protected/views
   ```

3. If you have a link for a previous project, remove it from the `webroot` directory.

   ```
   rm /opt/lampp/htdocs/cddb
   ```

4. Create a link in the `webroot` directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   sudo ln -s ~/projects/ch3 cbdb
   ```

5. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`) and configure for Yii development with PHPUnit.

6. Create a database named `cbdb` and load the database schema (`~/projects/ch3/protected/data/schema.sql`) into it.

7. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans it is `ch3/Source Files/protected/config/main.php`).

Adding a User Object with CRUD

As a foundation for our user management system, we will add a User table to the database and then use Gii to build a quick functional interface.

Engage Thrusters

1. Let's set the first building block by adding a User table containing the following information:

   ❏ A username

   ❏ Password hash

   ❏ Reference to a person entry for first name and last name

   In NetBeans, open a SQL Command window for the `cbdb` database and run the following command:

```
CREATE TABLE 'user' (
  'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
  'username' varchar(20) NOT NULL,
  'pwd_hash' char(34) NOT NULL,
  'person_id' int(10) unsigned NOT NULL,
  PRIMARY KEY ('id'),
  UNIQUE KEY 'username' ('username'),
  CONSTRAINT 'userperson_ibfk_2' FOREIGN KEY
    ('person_id') REFERENCES 'person' ('id') ON DELETE
    CASCADE
) ENGINE=InnoDB;
```

2. Open a web browser to the Gii URL `http://localhost/cbdb/index.php/gii` (the password configured in the sample code is `yiibook`) and use Gii to generate a model from the user table. Refer to the *Generating an Application Scaffold* section in *Project 1*, *Develop a Comic Book Database*, for a more detailed description of how to use Gii.

3. Then, use Gii to generate CRUD from the user model.

4. Back in NetBeans, add a link to the user index in your site's logged in menu (**ch3 | Source Files | protected | views | layouts | main.php)**. It should look like this:

```
} else {
  $this->widget('zii.widgets.CMenu',array(
    'activeCssClass' => 'active',
    'activateParents' => true,
    'items'=>array(
      array('label'=>'Home',
        'url'=>array('/site/index')),
      array('label'=>'Comic Books',
        'url'=>array('/book'),
          'items' => array(
            array('label'=>'Publishers',
              'url'=>array('/publisher')),
          )
        ),
    array('label'=>'Users',
      'url'=>array('/user/index')),
      array('label'=>'Logout ('.Yii::app()->user-
        >name.')', 'url'=>array('/site/logout'))
    ),
  ));
}
?>
```

5. Right-click on the project name, run the site, and log in with the default username and password (`admin`/`admin`). You will see a menu that includes a link named **Users**.
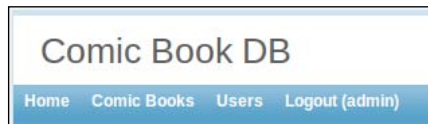
6. If you click on the **Users** link in the menu and then click on **Create User**, you will see a pretty awful-looking user-creation screen. We are going to fix that. First, we will update the user form to include fields for first name, last name, password, and repeat password. Edit **ch3 | Source Files | protected | views | user | _form.php** and add those fields.

7. Start by changing all instances of `$model` to `$user`. Then, add a call to `errorSummary` on the person data under the `errorSummary` call on user.

```
<?php echo $form->errorSummary($user); ?>
<?php echo $form->errorSummary($person); ?>
```

8. Add rows for first name and last name at the beginning of the form.

```
<div class="row">
  <?php echo $form->labelEx($person,'fname'); ?>
  <?php echo $form->textField($person,'fname',array
    ('size'=>20,'maxlength'=>20)); ?>
  <?php echo $form->error($person,'fname'); ?>
</div>

<div class="row">
  <?php echo $form->labelEx($person,'lname'); ?>
  <?php echo $form->textField($person,'lname',array
    ('size'=>20,'maxlength'=>20)); ?>
  <?php echo $form->error($person,'lname'); ?>
</div>
```

9. Replace the `pwd_hash` row with the following two rows:

```
<div class="row">
  <?php echo $form->labelEx($user,'password'); ?>
  <?php echo $form->passwordField($user,'password',array
    ('size'=>20,'maxlength'=>64)); ?>
  <?php echo $form->error($user,'password'); ?>
</div>

<div class="row">
  <?php echo $form->labelEx($user,'password_repeat'); ?>
  <?php echo $form->passwordField($user,'password_repeat',array
    ('size'=>20,'maxlength'=>64)); ?>
  <?php echo $form->error($user,'password_repeat'); ?>
</div>
```

10. Finally, remove the row for `person_id`.

11. These changes are going to completely break the User create/update form for the time being.

    We want to capture the password data and ultimately make a hash out of it to store securely in the database. To collect the form inputs, we will add password fields to the User model that do not correspond to values in the database. Edit the User model **ch3** | **Source Files** | **protected** | **models** | **User.php** and add two public variables to the class:

```
class User extends CActiveRecord
{
        public $password;
        public $password_repeat;
```

83

12. In the same User model file, modify the attribute labels function to include labels for the new password fields.

```
public function attributeLabels()
{
  return array(
    'id' => 'ID',
    'username' => 'Username',
    'password' => 'Password',
    'password_repeat' => 'Password Repeat'
  );
}
```

13. In the same User model file, update the rules function with the following rules:

   ❑ Require username

   ❑ Limit length of username and password

   ❑ Compare password with password repeat

   ❑ Accept only safe values for username and password

   We will come back to this and improve it, but for now, it should look like the following:

```
public function rules()
{
  // NOTE: you should only define rules for those attributes
  //that will receive user inputs.
    return array(
      array('username', 'required'),
      array('username', 'length', 'max'=>20),
      array('password', 'length', 'max'=>32),
      array('password', 'compare'),
      array('password_repeat', 'safe'),
    );
}
```

14. In order to store the user's first and last name, we must change the `Create` action in the User controller **ch3** | **Source Files** | **protected** | **controllers** | **UserController. php** to create a Person object in addition to a User object.

Change the variable name `$model` to `$user`, and add an instance of the Person model.

```
public function actionCreate()
{
  $user=new User;
  $person=new Person;

  // Uncomment the following line if AJAX validation is
  //needed
  // $this->performAjaxValidation($user);

  if(isset($_POST['User']))
  {
    $user->attributes=$_POST['User'];
    if($user->save())
      $this->redirect(array('view','id'=>$user->id));
  }

  $this->render('create',array(
    'user'=>$user,
    'person'=>$person,
  ));
}
```
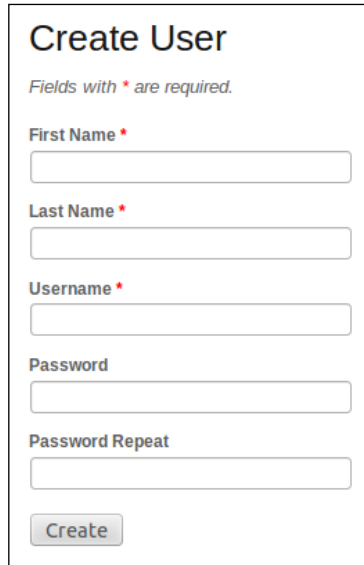
15. Don't reload the **create user** page yet. First, update the last line of the User Create view **ch3 | Source Files | protected | views | user | create.php** to send a User object and a Person object.

```
<?php echo $this->renderPartial('_form',
  array('user'=>$user, 'person' =>$person)); ?>
```

16. Make a change to the `attributeLabels` function in the Person model (**ch3 | Source Files | protected | models | Person.php**) to display clearer labels for first name and last name.

```
public function attributeLabels()
{
  return array(
    'id' => 'ID',
    'fname' => 'First Name',
    'lname' => 'Last Name',
  );
}
```

The resulting user form should look like this:

Create User

*Fields with * are required.*

First Name *

Last Name *

Username *

Password

Password Repeat

Create

17. Looks pretty good, but if you try to submit the form, you will receive an error. To fix this, we will change the User Create action in the User controller **ch3** | **Source Files** | **protected** | **controllers** | **UserController.php** to check and save both User and Person data.

```
if(isset($_POST['User'], $_POST['Person']))
  {
    $person->attributes=$_POST['Person'];
    if($person->save()) {
      $user->attributes=$_POST['User'];
      $user->person_id = $person->id;
      if($user->save())
        $this->redirect(array('view','id'=>$user->id));
    }
  }
```

18. Great! Now you can create users, but if you try to edit a user entry, you see another error. This fix will require a couple of more changes.

   First, in the user controller **ch3** | **Source Files** | **protected** | **controllers** | **UserController.php**, change the `loadModel` function to load the user model with its related person information:

```
$model=User::model()
  ->with('person')
  ->findByPk((int)$id);
```

19. Next, in the same file, change the `actionUpdate` function. Add a call to save the person data, if the user save succeeds:

```
if($model->save()) {
   $model->person->attributes=$_POST['Person'];
   $model->person->save();
   $this->redirect(array('view','id'=>$model->id));
}
```

20. Then, in the user update view **ch3 | Source Files | protected | views | user | update.php**, add the person information to the form render.

```
<?php echo $this->renderPartial('_form',
   array('user'=>$model, 'person' => $model->person)); ?>
```

21. One more piece of user management housekeeping; try deleting a user. Look in the database for the user and the person info. Oops. Didn't clean up after itself, did it? Update the User controller **ch3 | Source Files | protected | controllers | UserController.php** once again. Change the call to delete in the User delete action:

```
$this->loadModel($id)->person->delete();
```

Objective Complete - Mini Debriefing

We have added a new object, User, to our site, and associated it with the Person object to capture the user's first and last name. Gii helped us get the basic structure of our user management function in place, and then we altered the model, view, and controller to bring the pieces together.

Making a User Management Interface

The default Yii object index provides a nice summary listing of the user entries, but for many applications, it is more efficient to have a quick search capability. For this, Yii provides an additional "admin" view. We are going to completely replace the default listing with the admin view and update the scaffold view with a better integration of User with Person information for searching and sorting.

Engage Thrusters

1. Delete the file **ch3 | Source Files | protected | views | user | index.php**.
2. Rename the file **ch3 | Source Files | protected | views | user | admin.php** to `index.php`.
3. In the files `create.php`, `update.php`, and `view.php` in **ch3 | Source Files | protected | views | user**, remove the following line from the menu array:

```
array('label'=>'Manage User', 'url'=>array('admin')),
```

4. In the User controller **ch3** | **Source Files** | **protected** | **controllers** | **UserController. php**, delete the function named `actionIndex`.

5. Also in the User controller, remove the admin `accessRule` for the `admin` action. The admin `accessRule` should look like the following:

```
array('allow',
  // allow admin user to perform 'delete' actions
  'actions'=>array('delete'),
  'users'=>array('admin'),
),
```

Also, change the redirect in the delete action to send to the index.

```
$this->redirect(isset($_POST['returnUrl']) ?
  $_POST['returnUrl'] : array('index'));
```

6. In the same file, rename the function `actionAdmin` to `actionIndex`, and change the call to render in the newly renamed `actionIndex` function to render to `index` instead of `admin`. Now, if you click on the **Users** link in the menu, you will see a user management grid, instead of a list of user entries.

However, the information in the grid could be more useful.

7. Edit the new user index **ch3** | **Source Files** | **protected** | **views** | **user** | **index.php**. Remove the unnecessary column values `id`, `pwd_hash`, and `person_id` from the view. Add the columns we do want to see, namely first name and last name. These fields come from a related object, so their entries will look a little different. The file should look as follows:

```
'columns'=>array(
  'username',
  array(
    'name' => 'person_fname',
    'header' => 'First Name',
    'value' => '$data->person->fname',
  ),
  array(
    'name' => 'person_lname',
    'header' => 'Last Name',
    'value' => '$data->person->lname',
  ),
  array(
    'class'=>'CButtonColumn',
  ),
),
```

The entries for first name and last name include:

- ❑ A name value, which is the name of the search variable
- ❑ A header value, which is the column label
- ❑ A data value, which is the data field that will populate the column output

8. Edit the user model **ch3** | **Source Files** | **protected** | **models** | **User.php**, and add public variables to catch the search fields `person_fname` and `person_lname`:

```
class User extends CActiveRecord
{
  public $password;
  public $password_repeat;
  public $person_fname;
  public $person_lname;
```

9. In the same file, add a search field entry to the rules function with username, `person_fname`, and `person_lname`:

```
public function rules()
{
  // NOTE: you should only define rules for
  //those attributes that will receive user inputs.
  return array(
    array('username', 'required'),
    array('username', 'length', 'max'=>20),
    array('password', 'length', 'max'=>32),
    array('password', 'compare'),
    array('password_repeat', 'safe'),
    array('username, person_fname, person_lname', 'safe',
    'on'=>'search'),
  );
}
```

10. The search function will require the most changes. We will need to remove the unused fields (`id`, `pwd_hash`, and `person_id`), update the username field to indicate that it is from the base model, add the person relationship to the criteria, and add comparisons of the related fields (first and last name).

```
public function search()
{
  $criteria=new CDbCriteria;

  $criteria->compare('t.username',$this-
    >username,true);
  $criteria->compare('person.fname',$this-
    >person_fname,true);
  $criteria->compare('person.lname',$this-
    >person_lname,true);

  $criteria->with = array('person');

  return new CActiveDataProvider($this, array(
    'criteria'=>$criteria,
  ));
}
```

Now the grid will display all of the fields perfectly.

However, you can only sort on the username column.

11. In the User model search function, add a sort object with first name and last name fields and include it in the data provider to activate sort on the first name and last name columns.

```
$sort = new CSort;
$sort->attributes = array(
  'person_fname' => array(
    'asc' => 'person.fname',
    'desc' => 'person.fname DESC',
  ),
  'person_lname' => array(
    'asc' => 'person.lname',
    'desc' => 'person.lname DESC',
  ),
  '*',
);

return new CActiveDataProvider($this, array(
    'criteria'=>$criteria,
 'sort' => $sort,
));
```

12. Oh! One more thing. Have you clicked on the **Advanced Search** link yet?

That doesn't look great.

We can clean up the advanced search form **ch3 | Source Files | protected | views | users | _search.php**. Remove ID and password hash fields, and add first name and last name fields.

```
<div class="row">
  <?php echo $form->label($model,'username'); ?>
  <?php echo $form->textField($model,'username',array
    ('size'=>20,'maxlength'=>20)); ?>
</div>

<div class="row">
  <?php echo $form->label($model,'First Name'); ?>
  <?php echo $form->textField($model,'person_fname',array
    ('size'=>10,'maxlength'=>10)); ?>
</div>

<div class="row">
  <?php echo $form->label($model,'Last Name'); ?>
  <?php echo $form->textField($model,'person_lname',array
    ('size'=>10,'maxlength'=>10)); ?>
</div>
```

Now it looks good!

13. The last view we will change is named View. Edit **ch3** | **Source Files** | **protected** | **views** | **users** | **view.php**. Delete `id`, `pwd_hash`, and `person_id` from the attributes array. Add `person_fname` and `person_lname` to the list.

```
array(
  'name' => 'person_fname',
  'header' => 'First Name',
  'value' => $model->person->fname,
),
array(
  'name' => 'person_lname',
  'header' => 'Last Name',
  'value' => $model->person->lname,
),
```

Objective Complete - Mini Debriefing

We have removed the original user index and replaced it with the admin page that Yii provides, but modified so that the information is relevant, searchable, and sortable.

Classified Intel

We could make the index even more compact by moving the **Create User** link into the page, perhaps incorporating it into the grid, and removing the side menu, because the only other link is back to the index, and changing the layout to one column, instead of two.
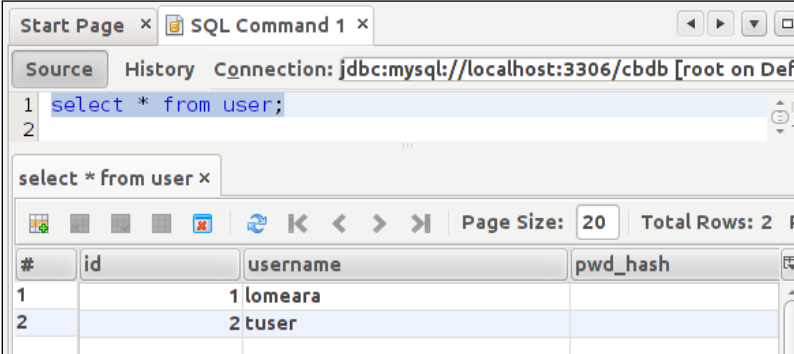
Storing Passwords

In this task, we will add a hashing function and store the hashed password values in the database.

Engage Thrusters

1. We have a nice user management interface, but if you open a SQL command window and query the user table, you will see that the password field for each user is empty.

   ```
   select * from user;
   ```
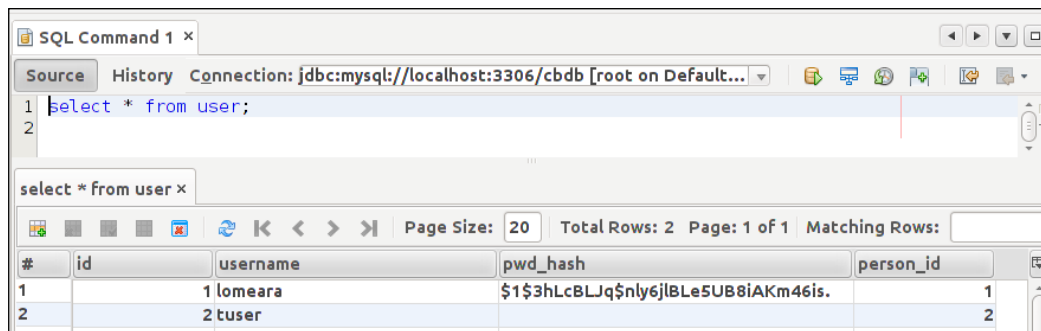

2. We need to store the password, and in order to do that, we need to make a function to hash passwords. We will implement this function in the User model and do it in a rather simplistic way, using the crypt library that comes with PHP and providing no salt value, so that it is randomly generated by the library. You can replace this function with your own preferred method of hashing.

   ```php
   public function hash($value)
   {
     return crypt($value);
   }
   ```

3. Next, we need to call the encryption function whenever we store a password – on create and on update – so we will overload the `beforeSave` function to do the hashing. Add the following function to the User model:

```
protected function beforeSave()
{
  if (parent::beforeSave())
    $this->pwd_hash = $this->hash($this->password);
  return true;
}
  return false;
}
```

Now, if you add or update a user and look at the user table, you will see a hash value in your user entry.

4. In preparation for logging in, let's go ahead and add a function to check a password value against the hashed value.

```
public function check($value)
{
  $new_hash = crypt($value, $this->pwd_hash);
  if ($new_hash == $this->pwd_hash) {
    return true;
  }
  return false;
}
```

Objective Complete - Mini Debriefing

We have added a hashing function to our User model to perform one way hashing on password values, then applied the hashing function to password values after they have been validated. We prepared for the next step by adding a hash check function to the User model as well. At this point, the hashing will not be applied to the login, but in the next task, we will activate it.

Activating Database User Login

In this task, we will convert the login action from the default authentication system provided by Yii to the authentication we have prepared in the previous tasks.

Prepare for Lift Off

We are about to cut over to a new authentication system. Before we do, be sure to create a user for yourself with a password that you know! If you haven't already, log in as `admin`/`admin`, go to the **Users** screen, create a user named `admin` with a password `test`. You can give this user whatever first and last name you like. We are about to use it to log in.

Engage Thrusters
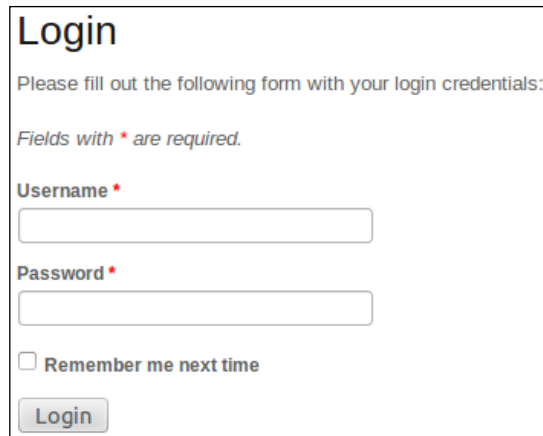
1. Edit the `UserIdentity` file **ch3** | **Source Files** | **protected** | **components** | **UserIdentity.php** and replace the contents of the authenticate function with the following:

```
$user = User::model()
->findByAttributes(array(
    'username' => $this->username
));

if($user===null)
  $this->errorCode=self::ERROR_USERNAME_INVALID;
else
if ($user->check($this->password))
  {
    $this->errorCode=self::ERROR_NONE;
  }
else
  $this->errorCode=self::ERROR_PASSWORD_INVALID;
return !$this->errorCode;
```

Now, the authenticate function will look in the database for the provided username. If that user is found, it will check the provided password against the user's password hash.

2. Give it a try. Log out (if you are logged in), and try logging back in as `admin`/`admin`. Now log in with the `admin` user we created earlier (`admin`/`test`).

3. Before we forget, edit the login view **ch3** | **Source Files** | **protected** | **views** | **site** | **login.php** and remove the demo and `admin` user hint.

Login

Please fill out the following form with your login credentials:

*Fields with * are required.*

Username *

Password *

☐ Remember me next time

Login

Objective Complete - Mini Debriefing

Now, instead of having to edit the `UserIdentity` file, and hardcode in another user/password combination, you can use the web interface to create as many users as you like. If there is a user who no longer needs access to the system, you can delete the user and his/her credentials will no longer work. This approach will be much easier to maintain.

Enforcing Secure Passwords

Looking again at user creation, we can see another problem. You can create a user with no password. That is not so bad, because the login form requires a password. If your user has no password, he will not be able to login, but what about the quality of the passwords? If you try to enter a one-character password, no problem, you can do it. This might be ok if you are the only person creating users and entering passwords. You can be careful to give your users passwords that are difficult to guess. You can devise and enforce your own password strength requirements, but typically, sooner or later, you are going to let your users set their own passwords. When this happens, you will want to enforce some checking to make sure the passwords your users set are difficult to guess. Otherwise, your users and your site are vulnerable to password cracking.

You can use this basic pattern for applying a password strength scheme and implement your own password strength requirements that are appropriate to your site. We will go with a basic requirement of a minimum length of eight characters, including at least one capital character, at least one number, and at least one non-alphanumeric character.

This pattern is also useful for implementing any custom validation rule.

Engage Thrusters

1. Open the user model file for edit (**ch3** | **Source Files** | **protected** | **models** | **User.php**).

2. Add a function named `passwordStrengthOk` as follows:

```
public function passwordStrengthOk($attribute, $params)
{
  // default to true
  $valid = true;

  // at least one number
  $valid = $valid && preg_match
    ('/.*[\d].*/', $this->$attribute);

  // at least one non-word character
  $valid = $valid && preg_match
    ('/.*[\W].*/', $this->$attribute);

  // at least one capital letter
  $valid = $valid && preg_match
    ('/.*[A-Z].*/', $this->$attribute);

  if (!$valid)
    $this->addError($attribute, "Does not meet password
      requirements.");

  return $valid;
}
```

3. Add two new rules to the validation array:

   ❑ Require a unique username.

   ❑ Add the new rule we created, `passwordStrengthOk` and change one old rule. Add a check for minimum password length of 8 to the password length requirements.

```
return array(
  array('username', 'required'),
  array('username', 'unique'),
```

```
    array('password, password_repeat', 'required'),
    array('username', 'length', 'min' => 3, 'max'=>20),
    array('password', 'length', 'min' => 8, 'max'=>32),
    array('password', 'compare', 'compareAttribute' =>
      'password_repeat'),
    array('password', 'passwordStrengthOk'),
    array('username, password, password_repeat', 'safe'),
    array('username, person_fname, person_lname', 'safe',
      'on'=>'search'),
  );
```

4. But what if we want to update something about the user, such as change the username, and not enter a new password? To do this, we will use a scenario. First, update the rules that apply to passwords and add the scenario parameter, so that the rules are only applied when the scenario is in play.

```
return array(
    array('username', 'required'),
    array('username', 'unique'),
    array('password, password_repeat', 'required', 'on' =>
      'passwordset'),
    array('username', 'length', 'min' => 3, 'max'=>20),
    array('password', 'length', 'min' => 8, 'max'=>32, 'on' =>
      'passwordset'),
    array('password', 'compare', 'compareAttribute' =>
      'password_repeat'),
    array('password', 'passwordStrengthOk', 'on' =>
      'passwordset'),
    array('username, password, password_repeat', 'safe'),
    array('username, person_fname, person_lname', 'safe',
      'on'=>'search'),                    );
```

   i. Then, in the User controller, activate the `passwordset` scenario whenever we want the `passwordset` rules to apply. In the `Create` function, we always want the scenario to apply, so pass it to the model constructor at the start.

```
$user=new User('passwordset');
```

   ii. In the `Update` function, we only want to apply the scenario when a password field has been entered, so set the scenario on the model conditionally. After the attributes are gathered from the form, if the password or `password_repeat` value has been set, apply the scenario.

```
$model->attributes=$_POST['User'];
if ($model->password || $model->password_repeat)
  $model->scenario = 'passwordset';
```

99

5. Let's make sure we did all of that correctly by making and running a functional test.

6. First, we will augment our testing setup by downloading the Selenium standalone server from `http://seleniumhq.org/`.

7. Then, update the `phpunit` config to define the browsers that you will test against. In our example, we will test against Firefox (of course, you must have Firefox installed to do this). Add the following section to **ch3 | Test Files | phpunit.xml**:

```xml
<selenium>
      <browser name="Firefox" browser="*firefox" />
</selenium>
```

8. Open **ch3 | Test Files | WebTestCase.php** and change `TEST_BASE_URL` to the URL of our site.

```php
define('TEST_BASE_URL', 'http://localhost/cbdb/');
```

9. Start the Selenium standalone server by opening a terminal window, changing to the directory where you downloaded the standalone server, and running the following command (updated to include the version of the server you downloaded):

```
java -jar selenium-server-standalone-<version-number>.jar
```

10. Navigate to **ch3 | Test Files**.

11. Right-click on the folder named **functional** and select **New | PHP File**.

12. Enter `UserTest` for the filename and click on `Finish`.

13. Input the following contents into the new test file.

```php
<?php
class UserTest extends WebTestCase {

  protected function setUp() {
    parent::setUp();

    $this->start();
    $this->open('');

    // login
    $this->clickAndWait('link=Login');
    $this->type('name=LoginForm[username]','admin');
    $this->click("//input[@value='Login']");
    $this->waitForTextPresent
      ('Password cannot be blank.');
    $this->type('name=LoginForm[password]','test');
    $this->clickAndWait("//input[@value='Login']");
```

```php
      // go to users page
      $this->clickAndWait('link=Users');
    }

    public function testPasswordMatch() {
      $this->clickAndWait('link=Create User');
      $this->type('name=Person[fname]','Func');
      $this->type('name=Person[lname]','Test');
      $this->type('name=User[username]','functest');
      $this->type('name=User[password]','functest');
      $this->type('name=User
        [password_repeat]','nomatchpass');
      $this->clickAndWait("//input[@value='Create']");
      $this->assertTextPresent
        ('Password must be repeated exactly.');
      $this->assertTextPresent
        ('Does not meet password requirements.');
      $this->assertTextNotPresent
        ('Password is too short
        (minimum is 8 characters).');
    }

    public function testPasswordTooShort() {
      $this->clickAndWait('link=Create User');
      $this->type('name=Person[fname]','Func');
      $this->type('name=Person[lname]','Test');
      $this->type('name=User[username]','functest');
      $this->type('name=User[password]','moo');
      $this->type('name=User[password_repeat]','moo');
      $this->clickAndWait("//input[@value='Create']");
      $this->assertTextPresent
        ('Password is too short
        (minimum is 8 characters).');
      $this->assertTextPresent
        ('Does not meet password requirements.');
      $this->assertTextNotPresent
        ('Password must be repeated exactly.');
    }

    public function testGoodPassword() {
      $this->clickAndWait('link=Create User');
      $this->type('name=Person[fname]','Func');
      $this->type('name=Person[lname]','Test');
      $this->type('name=User[username]','functest');
```

```
        $this->type('name=User[password]','m00!Isay');
        $this->type('name=User[password_repeat]','m00!Isay');
        $this->clickAndWait("//input[@value='Create']");
        $this->assertTextPresent('View User');
    }

    public function testDeleteUser() {
        $this->clickAndWait
          ("xpath=(//img[@alt=\"View\"])[2]");
        $this->clickAndWait('link=Delete User');
        $this->assertConfirmation
          ('Are you sure you want to delete this item?');
        $this->assertTextNotPresent('functest');
    }
}
?>
```

14. Be sure to save the new contents. Then, while viewing the new test file in NetBeans, press *Shift + F6* to run the functional test. You should see Selenium and Firefox windows flash on your screen as the tests run.

The tests should complete successfully and confirm that the password validation rules are applied correctly.

Objective Complete - Mini Debriefing

In order to improve our site security, we have added a custom validation rule to the user model. The new rule implements a password strength requirement that we defined, but you can replace this with your own custom definition or an existing library, such as CrackLib. To make sure your new rule is being enforced correctly, and to demonstrate functional testing with Selenium, we added a set of Selenium tests.

Adding User Functions – Wishlist

To demonstrate access control, we will create a new function for users of our site to show them our comic book wishlist. When a special occasion is coming up, your friends and family will be able to log in and view your wishlist to get gift ideas.

Engage Thrusters

1. Start by adding a new table to the database as follows:

```
CREATE TABLE 'wish' (
  'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
  'title' varchar(256) NOT NULL,
  'issue_number' varchar(10) DEFAULT NULL,
  'type_id' int(10) unsigned DEFAULT NULL,
  'publication_date' date DEFAULT NULL,
  'store_link' varchar(255) DEFAULT NULL,
  'notes' text DEFAULT NULL,
  'got_it' int(10) unsigned DEFAULT NULL,
  PRIMARY KEY ('id'),
  KEY 'type_id' ('type_id'),
  KEY 'got_it' ('got_it'),
  CONSTRAINT 'wish_ibfk_1' FOREIGN KEY ('type_id')
    REFERENCES 'type' ('id'),
  CONSTRAINT 'wish_ibfk_2' FOREIGN KEY ('got_it')
    REFERENCES 'user' ('id')
) ENGINE=InnoDB;
```

2. Add join tables for author, illustrator, and publisher as follows:

```
CREATE TABLE 'wishauthor' (
  'wish_id' int(10) unsigned NOT NULL,
  'author_id' int(10) unsigned NOT NULL,
  PRIMARY KEY ('wish_id','author_id'),
  KEY 'author_id' ('author_id'),
  CONSTRAINT 'wishauthor_ibfk_1' FOREIGN KEY ('wish_id')
    REFERENCES 'wish' ('id') ON DELETE CASCADE,
  CONSTRAINT 'wishauthor_ibfk_2' FOREIGN KEY
    ('author_id') REFERENCES 'person' ('id')
) ENGINE=InnoDB;
CREATE TABLE 'wishillustrator' (
  'wish_id' int(10) unsigned NOT NULL,
  'illustrator_id' int(10) unsigned NOT NULL,
  PRIMARY KEY ('wish_id','illustrator_id'),
  KEY 'illustrator_id' ('illustrator_id'),
  CONSTRAINT 'wishillustrator_ibfk_1' FOREIGN KEY
    ('wish_id') REFERENCES 'wish' ('id') ON DELETE
    CASCADE,
```

```
      CONSTRAINT 'wishillustrator_ibfk_2' FOREIGN KEY
        ('illustrator_id') REFERENCES 'person' ('id')
  ) ENGINE=InnoDB;

  CREATE TABLE 'wishpublisher' (
    'wish_id' int(10) unsigned NOT NULL,
    'publisher_id' int(10) unsigned NOT NULL,
    PRIMARY KEY ('wish_id','publisher_id'),
    KEY 'publisher_id' ('publisher_id'),
    CONSTRAINT 'wishpublisher_ibfk_1' FOREIGN KEY
      ('wish_id') REFERENCES 'wish' ('id') ON DELETE
      CASCADE,
    CONSTRAINT 'wishpublisher_ibfk_2' FOREIGN KEY
      ('publisher_id') REFERENCES 'publisher' ('id')
  ) ENGINE=InnoDB;
```

3.   The new tables and their relationships look like the following:
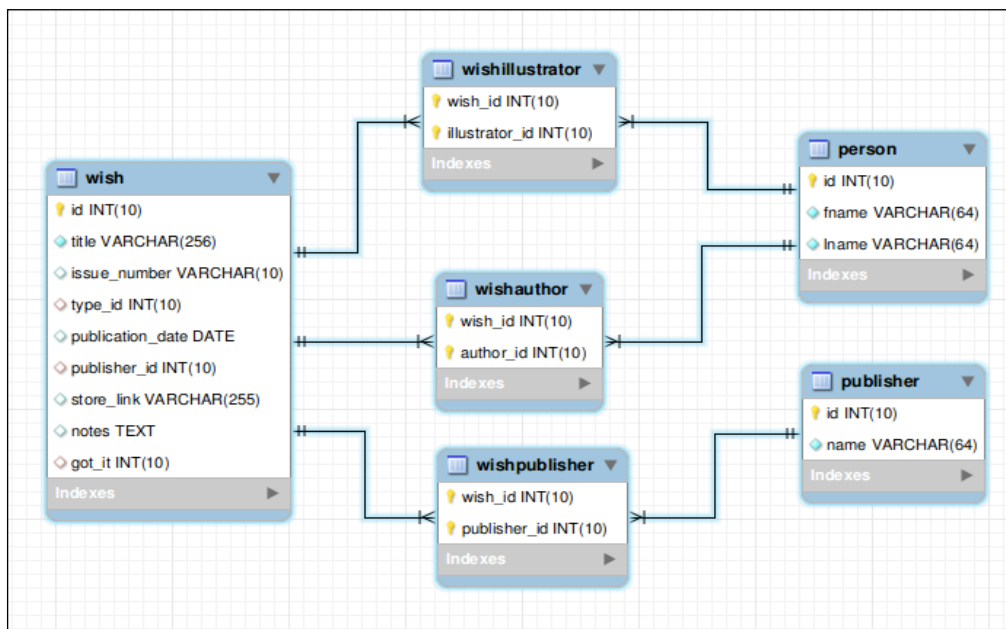
4.   Use Gii to generate a model and CRUD from the wish table, and models for all of the join tables (`wishauthor`, `wishillustrator`, and `wishpublisher`).

5. Add a new item to the Comic Book menu in **ch3** | **Source Files** | **protected** | **views** | **layouts** | **main.php** as follows:

```
array(
   'label'=>'Comic Books',
   'url'=>array('/book'),
    'items' => array(
     array('label'=>'Publishers',
       'url'=>array('/publisher')),
   array('label'=>'WishList', 'url'=>array('/wish/index')),
     )
  ),
```

Now, when you expand the Comic Books menu, you will see an entry for WishList, as follows:

6. Since wish has a lot of the same fields as book, we copied **ch3** | **Source Files** | **protected** | **views** | **book** | **_form.php** to the wish view folder, removed the unnecessary fields **value**, **price**, **signed**, **grade**, and **bagged**, and added the unique wish field, **store_link**. The resulting file **ch3** | **Source Files** | **protected** | **views** | **wish** | **_form.php** can be found in the chapter files **ch3** | **Source Files** | **example** | **wish** | **_form.php**. The form will not work until we make a few changes.

7. We have already pulled the author functions that you might want to use out of the Book controller and put them in a base controller that the Book controller is using. To access the author functions in the Wish controller, change the base class.

```
class WishController extends BController
```

   i. Delete the update action from the Wish controller and change the generated create action to the following:

```
   public function actionCreate()
 {
   $model=new Wish;
  $this->create($model);
 }
```

  ii. Create a function to record the association between a wish and an author.

```
protected function saveAssociation($model, $author)
 {
   // record wish/author association
    $wa = new WishAuthor;
   $wa->wish_id = $model->id;
   $wa->author_id = $author->id;
   $wa->save();
}
```

  iii. Add `removeAuthor` and `createAuthor` to the allowed actions for users. (We will adjust the permissions in a later task.)

```
array('allow', // allow authenticated user to perform
   'create' and 'update' actions
   'actions'=>array('create','update',
     'removeAuthor','createAuthor'),
   'users'=>array('@'),
),
```

8. Add support for authors to the Wish model **ch3** | **Source Files** | **protected** | **models** | **Wish.php**. Start by adding the following functions:

```
/*
* assign this author to this wish
*/
public function addAuthor($author) {
  $wishauthor = new WishAuthor();

  $author->save();
  $wishauthor->wish_id = $this->id;
  $wishauthor->author_id = $author->id;
  $wishauthor->save();
}

/*
* remove an author association from wish
*/
public function removeAuthor($author_id) {
  $pk = array('wish_id'=>$this->id, 'author_id' =>
    $author_id);
  WishAuthor::model()->deleteByPk($pk);
```

Then, add the following two relations:

```
'authors' => array(self::MANY_MANY, 'Person',
  'wishauthor(author_id, wish_id)', 'index'=>'id'),
  'wishauthors' => array(self::HAS_MANY, 'WishAuthor',
    'wish_id', 'index' => 'author_id'),
```

9. Add the author variable to the call to `renderPartial` in the Wish create and update views, **ch3 | Source Files | protected | views | wish | create.php** and **ch3 | Source Files | protected | views | wish | update.php** respectively. In both files, the call will look like the following:

```
<?php echo $this->renderPartial('_form',
  array('model'=>$model, 'author'=>$author)); ?>
```

Also include the author add JavaScript at the beginning of both files:

```
Yii::app()->clientScript->registerScriptFile(
  Yii::app()->request->baseUrl .
    '/js/book_form_ajax.js'
);
```

Objective Complete - Mini Debriefing

In this task, we have created another object, a wish. This object benefitted from its similarity to the book object and the work we had already done to associate books to authors. We will use this object to demonstrate configuring different capabilities for different users.

Configuring User Access

There is more than one way to define user access. One is the file-based method we replaced in this chapter. Another method is role-based and it is demonstrated in another project. For this project, we will define user-based access to the wishlist function, and we will provide two levels of access:

- Admin – our own login, which will be used to create and maintain the wishlist

- Everyone else – for our friends and family who want to view the wishlist and claim items that they have got for us

Prepare for Lift Off

To perform a cursory check (as opposed to a comprehensive suite of unit tests) of the changes we are about to make, you will need to have the ability to log in as two different users with different levels of access. We already have an `admin` user, which is our own login for creating and maintaining the wishlist. Create another account with username `guest` and password `Gu3st!!!` to test guest access. For our development and testing, this user represents all of our other users who will have the ability to view the wishlist and claim items they have got for us (so we don't get duplicate presents).

Run the following MySQL commands to insert some wish data into your database:

```
INSERT INTO 'wish' VALUES (1,'Moebius\' Airtight Garage
  Vol.1','1',1,'0000-00-00','http://www.amazon.com/
  Moebius-Airtight-Garage-Vol-1-No/dp/B00178YGFE/
  ref=sr_1_3?s=books&ie=UTF8&qid=1339476850&sr=1-
  3','',NULL),(2,'The Squiddy Avenger','1',1,'2012-06-
  21','www.amazon.com','',NULL),(3,'another great
  title','',1,'2012-06-21','','',NULL);

INSERT INTO 'person' VALUES
  (226,'Jean','Giraud'),(227,'John','Smith');
INSERT INTO 'wishauthor' VALUES (1,226),(2,227);
```

Engage Thrusters

As administrators, we pretty much have all the capabilities we need already. In fact, we may want to limit one thing. What good is our wishlist if we lose the surprise by seeing what our friends have got for us? Of course, you can cut out the parts that hide the information from `admin` or leave them in and find other ways to peek at your gifts in the database.

Let's start by limiting our guests' access, and then update the wishlist view to achieve the desired effect.

We are going to customize the users' menu view in a later task, so that for now, when we are logged in as a guest, we can easily click all options and see what we can and cannot do.

1. Log in as `guest`/`Gu3st!!!`.

2. From the menu, you can see and access the user index. Let's eliminate that option for our guests. For now, only the administrator will be able to view, create, update, or delete users. Open the user controller **ch3** | **Source Files** | **protected** | **controllers** | **UserController.php** and consolidate all of the actions into the allowed array for the `admin` user. When you are done, `accessRules` should look as follows:
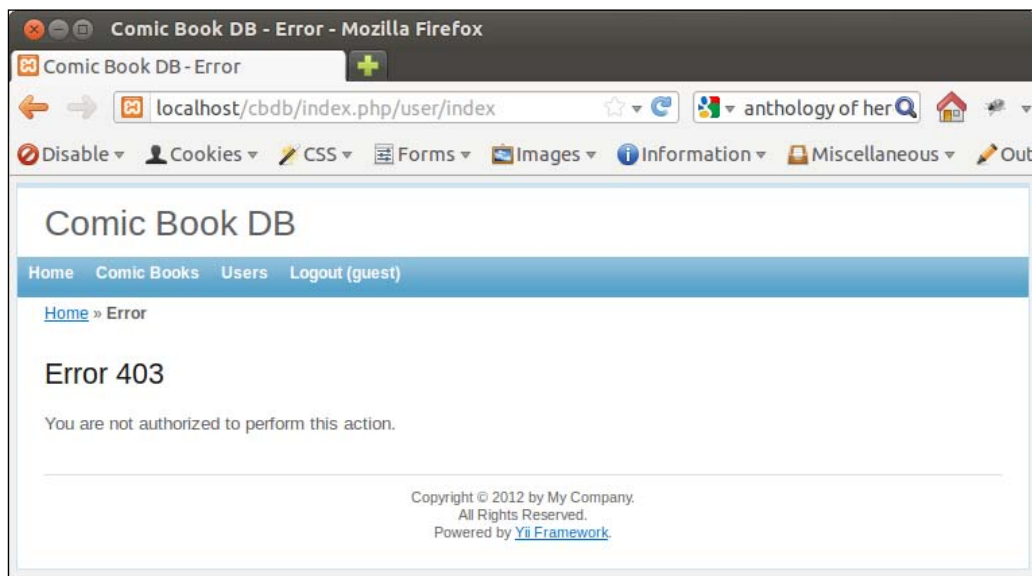
   ```
   public function accessRules()
   {
   ```

```
return array(
    array('allow', // allow admin user to perform 'delete'
      actions
        'actions'=>array('index', 'view', 'create', 'update',
          'delete'),
        'users'=>array('admin'),
    ),
    array('deny',  // deny all users
        'users'=>array('*'),
    ),
);
}
```

3. Now, click on **Users** in the menu and you should see an error message as shown in the following screenshot:

4. If you want to log out of guest and log back in as `admin` at this point to make sure you still have user management access, we don't blame you. Go ahead. Check it out. Then, log back in as `guest` to continue.

5. Guests really don't need to view publishers either. So make the same change to `accessRules` in **ch3** | **Source Files** | **protected** | **controllers** | **PublisherController.php**.

6. We want to share our list of books with our users, but not anonymous strangers. Also we don't want guests changing anything. To accomplish this, make the following changes to the book controller.

   Move access to index and view down to authenticated users.

   Move access to `create`, `update`, `removeAuthor`, and `createAuthor` down to `admin` user. Remove the all users section.

   The result should look as follows:

   ```php
   public function accessRules()
   {
     return array(
       array('allow', // allow authenticated user to
         perform 'index' and 'view' actions
         'actions'=>array('index','view'),
         'users'=>array('@'),
       ),
       array('allow', // allow admin user to perform
         'create' 'update' 'admin' and 'delete' actions
         'actions'=>array('create','update',
         'removeAuthor','createAuthor','admin','delete'),
         'users'=>array('admin'),
       ),
       array('deny',  // deny all users
       'users'=>array('*'),
     ),
                 );
   }
   ```

7. Wishlist is similar to the book list in the previous step, but we do want guests to be able to claim items from the list. Let's start by replacing the access rules in the Wish controller with the access rules we just made for Books.

   Now we will make one small side-track to make our wishlist look nice for our guests.

8. We created a field for a store link to make it convenient for our friends to click right on an item we want and purchase it. But right now that URL displays as un-clickable text. Edit **ch3 | Source Files | protected | views | wish | _view.php** and make this change to the `store_link` field to make it clickable from the index.

   ```php
   <b><?php echo CHtml::encode($data-
     >getAttributeLabel('store_link')); ?>:</b>
   <a href="<?php echo CHtml::encode($data->store_link);
     ?>" target="_blank">Purchase</a>
   <br />
   ```

9. We added `target="_blank"` to the link so that it will open in another browser window. That way our users can remain in our site.

10. Our users don't need to see ID values, and for that matter, neither do we. But they may want to click an item and read more about it. To that end, remove the ID field, and update the title field to be the link to the item.

```
<b><?php echo CHtml::encode($data-
  >getAttributeLabel('title')); ?>:</b>
<?php echo CHtml::link(CHtml::encode($data->title),
  array('view', 'id'=>$data->id)); ?>
<br />
```

11. Now, let's clean up the individual wish view **ch3** | **Source Files** | **protected** | **views** | **wish** | **view.php**. Replace the page title with the wish title.

```
<h1>Wish: <?php echo CHtml::encode($model->title);?></h1>
```

12. Remove `id` from the detail view attribute list, and change the store link to pass an attribute array, including a name, `type=raw`, and the defined link. We also want to display the type value as text instead of a number.

The final attributes of the detail view should look as follows:

```
'attributes'=>array(
  'title',
  'issue_number',
  array(
    'name' => 'type',
    'value' => $model->type->name,
  ),
  'publication_date',
  array(
    'name' => 'store link',
    'type' => "raw",
    'value' => "<a href=\"" . $model->store_link
    . "\" target=\"_blank\">Purchase</a>",
  ),
  'notes',
  'got_it',
),
```

The resulting wish details screen will look like the following:
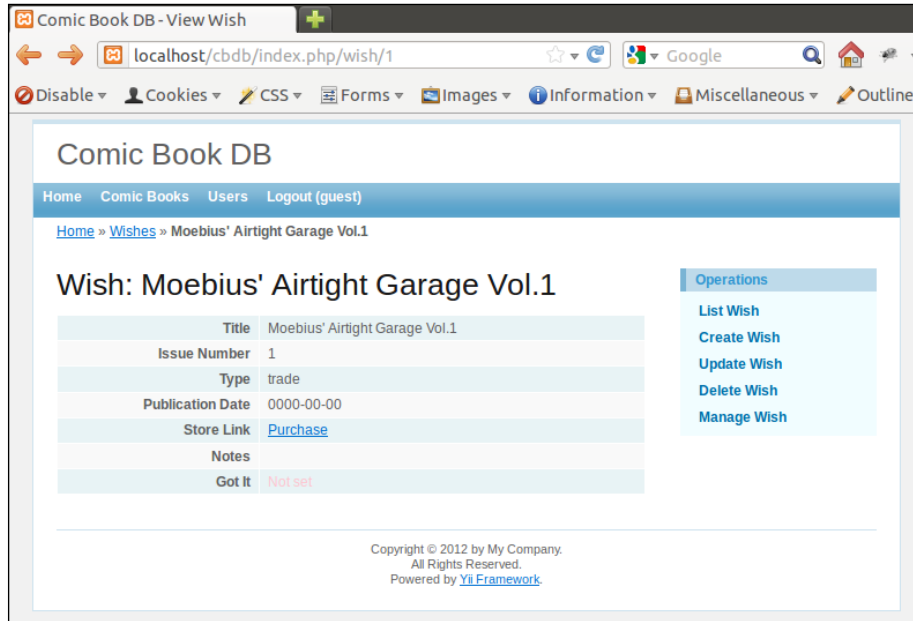
13. Now that the index and view are looking pretty nice for our guests, we should limit the wish list items so that a user only sees unclaimed items or items that user has claimed.

    First, we must extend the `UserIdentity` class to store and return the user's ID and name.

    i.   Open **ch3** | **Source Files** | **protected** | **components** | **UserIdentity.php** and add private fields named `_id` and `_username` to the class.

```
private $_id;
 private $_username;
```

    ii. Add functions to return the values.

```
public function getName()
 {
    return $this->_username;
}
 public function getId()
 {
    return $this->_id;
}
```

iii. Set the value when we have a successful authentication.

```
if ($user->check($this->password))
 {
  $this->_id = $user->id;
  $this->_username = $user->username;
  $this->errorCode=self::ERROR_NONE;
 }
```

14. Now that the user identity class can return the current user's ID and username, we can update the Index action in the Wish controller to limit the wish list, if the user is not `admin`.

```
public function actionIndex()
{
  if(Yii::app()->user->getName() == 'admin') {
    $dataProvider=new CActiveDataProvider('Wish');
  } else {
    $dataProvider=new CActiveDataProvider
      ('Wish', array(
        'criteria' => array(
          'condition' => 'got_it is null OR ' .
          'got_it=' . Yii::app()->user->getId(),
        )
      ));

    $this->render('index',array(
      'dataProvider'=>$dataProvider,
    ));
}
```

If you are currently logged in as a guest and try to access this page, you will see an error message. You will need to log out and log back in as a guest in order to access the Wish index.

At this point, to test our work, we logged in as `admin` and created a few wishes. Then we used the database `Execute Command` feature to set a few wish `got_it` values to various users, and left a few unclaimed. We logged in as each of those users to verify that we saw unclaimed wishes and the user's claimed wishes. We also logged in as `admin` to verify that `admin` sees the complete list.

15. The next part of the feature that we will need is the ability to claim a wish. To keep it a surprise, we are going to show this feature to users but not to `admin`.

Edit **ch3** | **Source Files** | **protected** | **views** | **wish** | **view.php** and change the attribute value for the field `got_it` as follows:

```
array(
    'name' => 'Got It',
    'value' => (Yii::app()->user->getName() != 'admin') ?
        $model->got_it : ''
),
```

16. Edit **ch3** | **Source Files** | **protected** | **views** | **wish** | **_view.php** and add the following to display a claim/unclaim checkbox:

```
<?php if (Yii::app()->user->getName() != 'admin') { ?>
    <b><?php echo CHtml::encode($data-
        >getAttributeLabel('got_it')); ?>:</b>
    <?php
        echo CHtml::checkBox("got",
            ($data->got_it == Yii::app()->user->getId()),
            array('class' => 'claim',
                'url'=> Yii::app()->controller->createUrl(
                    "claim",
                    array("id"=>$data->id)
                )
            )
        );
    ?>
<?php } ?>
<br />
```

The result should look like the following screenshot:
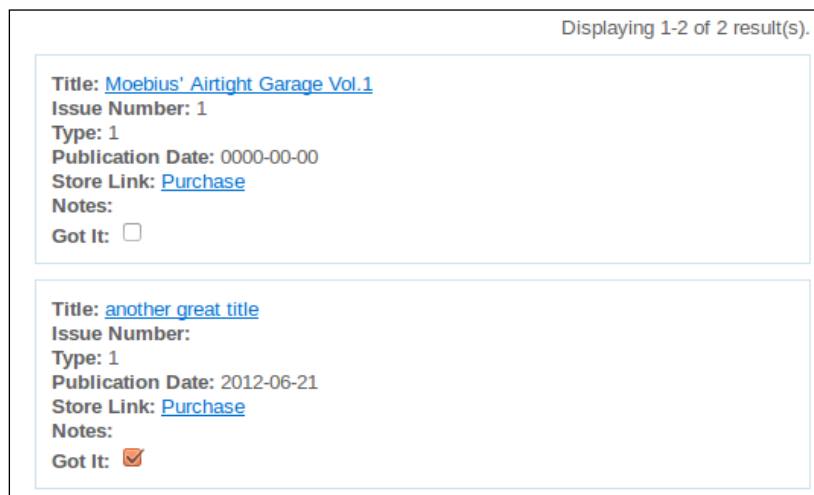
17. To support the checkbox toggle function we are going to add Ajax to catch the checkbox click. Create a file in **ch3 | Source Files | js named wish_list_ajax.js** with the following contents:

```
$(document).ready(function() {
  $('.claim').click(function() {
    $.ajax({
      type: 'get',
      url: $(this).attr('url'),
      data: {"ajax" : "1"},
      success: function(resp) {
        $("ul.authors").append(resp);
      },
      error: function() {
        alert("Error claiming wish.");
      }
    });
  });
});
```

18. To include the script in the wish index, edit **ch3 | Source Files | protected | views | wish | index.php** and add the script at the beginning of the file.

```
Yii::app()->clientScript->registerScriptFile(
   Yii::app()->request->baseUrl . '/js/wish_list_ajax.js'
);
```

19. Then, of course, we need to add the claim action itself to the Wish controller.

```
  public function actionClaim($id)
  {
    // request must be made via ajax
    if(isset($_GET['ajax'])) {
      $model=$this->loadModel($id);
      // if the wish was claimed by the user, toggle it
      //off
      if ($model->got_it == Yii::app()->user->getId()) {
        $model-> got_it = new CDbExpression('NULL');
      }
      // if the wish was claimed by no one, toggle it on
      if ($model->got_it == null) {
        $model->got_it = Yii::app()->user->getId();
      }
      $model->save();
    }
    else
      throw new CHttpException(400,'Invalid request.');
  }
```

Update the access rules to include the new action, allowing authenticated users to claim wishes.

```
array('allow', // allow authenticated user to perform
  'index' and 'view' actions
  'actions'=>array('index','view','claim'),
  'users'=>array('@'),
),
```

Objective Complete - Mini Debriefing

We have used the basic access control that is provided by Yii to limit general user access for viewing the comic book list and items and the wish list and items.

User Specific Menus

In the last task, we limited user access, but we did not update the menus. The site menu provides links to objects that users do not have permission to access, and the object menus provide links to actions that users do not have authorization to take. It would be nice if we could associate the menus to the access we have already defined, so that we do not have to manually coordinate menu contents with `accessRules`. To do this, we created an extension to the CMenu widget.

Engage Thrusters

1. Create a new file in **ch3** | **Source Files** | **protected** | **components** named `AuthMenu.php`.

2. Enter the following contents into the file:

```php
<?php

  Yii::import('zii.widgets.CMenu');

  /**
   * Auth Menu extends CMenu to apply access rules to
   menu items before
   * displaying them. The idea is to define menu items
   once and only
   * display relevant items.
   *
   * This extension was inspired by YiiSmartMenu
   *
   * @author Lauren O'Meara
   <lauren@plumflowersoftware.com>
```

```
 * @copyright Copyright &copy; 2012 Plum Flower
   Software
 * @version 0.1
 * @license New BSD Licence
 */
class AuthMenu extends CMenu
{
  public function init() {
    $this->items = $this->filterItems($this->items);
    return parent::init();
  }

  /**
   * Filter recursively the menu items received setting
     visibility true or
   * false according to controller/action preFilter
   *
   * @param array $items The menu items being filtered.
   * @return array The menu items with visibility
     defined by preFilter().
   */
  protected function filterItems(array $items){
    $app = Yii::app();
    foreach($items as $pos=>$item)
    {
      if(!isset($item['visible']))
      {
        // get the url parameter
        if(isset($item['url']) &&
          is_array($item['url']))
        $url=$item['url'][0];

        // parse the url into controller and action
        $parts = explode("/",$url);
        if ( count($parts) == 1) {
          $controller = $app->controller;
        $actionId = $parts[0];
      } else {
        $controllerId = ucfirst($parts[1]);
        $actionId = count($parts) > 2 ? $parts[2] :
          'index';
        $controllerList = $app-
          >createController($controllerId);
        $controller = $controllerList[0];
      }
```

```
        // generate a controller instance to access and
        //compare the rules
        $action = $controller->createAction($actionId);
        $filter = new CAccessControlFilter;
        $filter->setRules($controller->accessRules());
        $user = $app->getUser();
        $request = $app->getRequest();
        $ip = $request->getUserHostAddress();
        $item['visible'] = false;
        foreach ($filter->getRules() as $rule) {
          // we are making an assumption for now that all
          // menu items are GET actions
          if($rule->isUserAllowed($user, $controller,
            $action, $ip, 'GET') > 0) {
          $item['visible'] = true;
          break;
        }
      }
    }

    /**
     * If current item is visible and has sub items,
      loops recursively
     * on them.
     */
    if(isset($item['items']) && $item['visible'])
      $item['items']=$this->filterItems($item['items']);

    $items[$pos]=$item;
    }
  return $items;
  }
}
```

3. Next, consolidate the main site menu back into one widget call. Set the visible value for the `Home` item to `true`, and set the visible value for `Login` and `Logout` to conditional based on whether the user is logged into the site or not, using the `isGuest` command. Replace the call to CMenu in main layout file **ch3 | Source Files | protected | views | layouts | main.php**.

```
<div id="mainmenu">
  <?php
```

```
$this->widget('application.components.AuthMenu',array(
  'activeCssClass' => 'active',
  'activateParents' => true,
  'items'=>array(
    array('label'=>'Home', 'url'=>array('/site/index'),
    'visible' => true),
  array(
    'label'=>'Comic Books',
    'url'=>array('/book'),
    'items' => array(
      array('label'=>'Publishers',
        'url'=>array('/publisher')),
      array('label'=>'WishList',
        'url'=>array('/wish/index')),
    )
  ),
  array('label'=>'Users',
    'url'=>array('/user/index')),
  array('label'=>'Login', 'url'=>array('/site/login'),
    'visible'=>Yii::app()->user->isGuest),
  array('label'=>'Logout ('.Yii::app()->user->name.')',
    'url'=>array('/site/logout'), 'visible'=>!Yii::app()-
    >user->isGuest),
  ),
));
?>
</div><!-- mainmenu -->
```

The consolidated menu for `admin` should look the same as before, like the following screenshot:

The menu for a guest will have fewer items, as shown in the following screenshot:
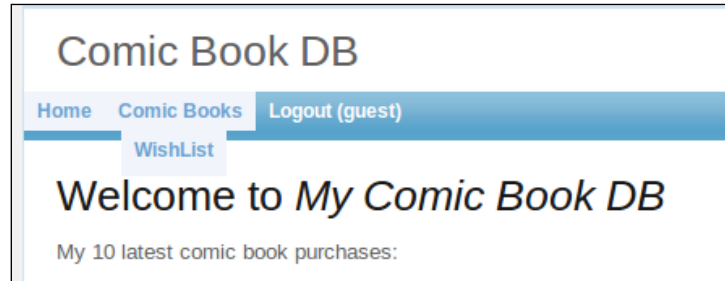
4. Also replace the call to CMenu in the two column layout **ch3** | **Source Files** | **protected** | **views** | **layouts** | **column2.php**.

```
$this->widget('application.components.AuthMenu', array(
    'items'=>$this->menu,
    'htmlOptions'=>array('class'=>'operations'),
));
```

The action menu on the right-hand side of the screen will now display only authorized actions. For `admin`, the list will include all actions, as shown in the following screenshot:

For guests, the list of authorized actions will not include `Create`, `Update`, or `Delete`, as shown in the following screenshot:
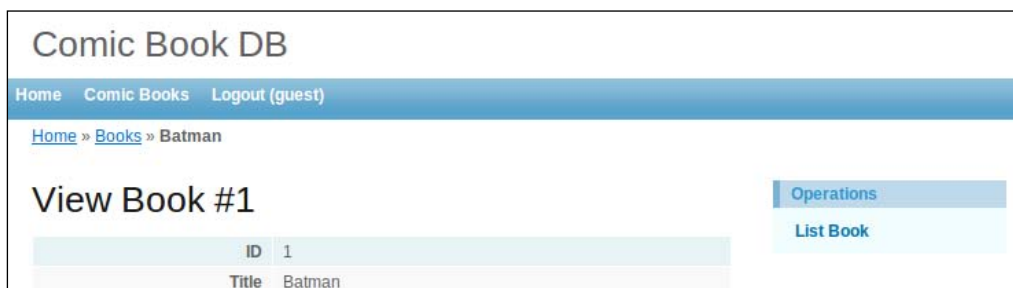
Objective Complete - Mini Debriefing

In the extension that we created, we iterate over the list of menu items and use `context` and the `url` parameter to determine which controller and action the menu item contains. Then we check the array in the `accessRules` function for the controller against the user and action to determine the visibility of the menu item.

Mission Accomplished

In this project, we have improved user management for our site by replacing the default file-based user management that Yii framework provides with database-stored users. By making this change, we get the benefit of the web interface to manage our users, instead of having to change the text in a source file.

We removed the default User index and replaced it with the Yii-generated admin page, which provides Ajax record searching and quick links to view/update/delete users. We also customized this view to include support for fields from the related table, `Person`. As a result we can search and sort on fields from `Person`, as well as `User`.

We improved site security by creating a custom validation rule that enforces some password strength requirements, and we apply this rule only when we need to change the password, not when we are making a change to an existing user.

We tested the implementation of this validation rule and tried out functional testing with Selenium.

We added a new function to demonstrate Yii access control settings. And we showed a way to display user-specific menus. This will get even better in the next project when we group users into roles.

Remember to review your security if you put this site online. You may want to review the following considerations:

- ▶ You may not want to use XAMPP for a public server
- ▶ If you do use XAMPP, at the least, enable secure mode
- ▶ Obtain and use an SSL certificate; require encrypted access to the login page, so that passwords are not transmitted unencrypted
- ▶ Definitely disable Gii in your Yii site configuration
- ▶ Also be sure to disable phpMyAdmin

You Ready to go Gung HO? A Hotshot Challenge

Here are some ideas to go gung ho with user functions:

- ▶ You could expand your wishlist system to allow other users to track their wishlists.
- ▶ You could extend the user object with a Boolean value to indicate active/inactive users, instead of deleting user entries.
- ▶ Extend the functional tests.
- ▶ Check out the Selenium Firefox extension and the Selenium IDE: PHP Formatters. Try using them to record more functional tests.
- ▶ Replace all index views with the admin view, like we did for Users.
- ▶ Add a function to transfer a wish into your book list, once you have received it.

Project 4

Level Up! Permission Levels

In this mission, we will implement a lending function and a fine grained access control. In our experience, most projects require the ability to define permissions at a very precise level. A good example of this is providing users with the ability to edit their own account information, but not the account information of other users. In this project, we will use Yii and available extensions to construct a custom permissions system for our comic book application.

Mission Briefing

We will add a library management page for you to manage the books you share and a lend/borrow page for your friends to see books that they can borrow and request. Then, we will replace the default Yii user access with a more extensive user management system that includes roles and access levels.

Why Is It Awesome?

Almost any web application you make is going to have users with different levels of access. This project will demonstrate a method for adding users and access control to any site you build with Yii. We will also touch on some website security issues here, but encourage you to study this topic well and enhance your knowledge of security with every site you build.

Your Hotshot Objectives

Here is an overview of the project steps:

- ▶ Adding Admin Function – Library Management
- ▶ Adding User Functions – Library
- ▶ Defining Roles and Access
- ▶ Adding the RBAC Extension
- ▶ Adding Roles to User Management
- ▶ Fine-tuning Permissions
- ▶ Making History

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project follow these steps, replacing the username lomeara with your own username:

1. Copy the project files into your working directory.

   ```
   cp –r ~/Downloads/project_files/Chapter\ 4/project_files
     ~/projects/ch4
   ```

2. Make the directories that Yii uses web writeable.

   ```
   cd ~/projects/ch4/
   ```

   ```
   sudo chown -R lomeara:www-data protected/runtime assets
     protected/models protected/controllers protected/views
   ```

3. Create a link in the `webroot` directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   ```

   ```
   sudo ln -s ~/projects/ch4 cbdb
   ```

4. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`) and configure for Yii development with PHPUnit.

5. Create a database named `cbdb` and load the database schema (`~/projects/ch4/protected/data/schema.sql`) into it.

6. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans, it is **ch4** | **Source Files** | **protected** | **config** | **main.php**).

> The admin login is admin/test.

Adding Admin Function – Library Management

Because we like to share books from our collection with our friends, we will add a lending function to our site. When a friend logs in, he/she will be able to see which books are available for borrowing, which he/she is currently borrowing, and which he/she has borrowed in the past.

Engage Thrusters

1. First, we will expand our comic book management interface with some new lending information. We will note which books we are willing to lend and while we are at it, let's make a field to track who is currently borrowing a book. Open an SQL command window for the `cbdb` database and run the following commands:

```
ALTER TABLE book ADD borrower_id int(10) unsigned default
  null;
ALTER TABLE book ADD lendable boolean default true;
ALTER TABLE book ADD FOREIGN KEY ('borrower_id')
  REFERENCES user(id);
```

2. Now we must expand the book model to include the new fields (**ch4** | **Source Files** | **protected** | **models** | **Book.php**).

  i. At the top of the model, add the new fields to the comments. This is not required, but it is good practice.

  ```
  * @property string $borrower_id
  * @property integer $lendable
  ```

  ii. Add a relation named `borrower` to connect the user who has borrowed the book, if `borrower_id` is set.

  ```
  * @property User $borrower
  ```

iii. Add `lendable` to the numeric field check in the rules function.

```
array('signed, bagged, lendable', 'numerical',
  'integerOnly'=>true),
```

iv. Add a check to make sure `borrower_id` has a maximum length of 10 in the `rules()` function.

```
array('type_id, value, price, grade_id, issue_number',
  'length', 'max'=>10),
```

The previous code then becomes:

```
array('type_id, value, price, grade_id, issue_number,
  borrowed_id', 'length', 'max'=>10),
array('type_id, value, price, grade_id, borrower_id',
  'length', 'max'=>10),
```

v. Add `lendable` to the searchable list.

```
array('id, title, type_id, publication_date, value,
  price, notes, signed, grade_id, bagged, issue_number,
  lendable', 'safe', 'on'=>'search'),
```

vi. Add entries in the `attributeLabels()` function.

```
'borrower_id' => 'Borrower',
'lendable' => 'Lendable',
```

vii. Add `lendable` to the compare criteria in the search function.

```
$criteria->compare('lendable',$this->lendable,true);
```

viii. Finally, add an entry named `borrower` for the `lent` field in the `relations` function.

```
'borrower' => array(self::BELONGS_TO, 'User',
  'borrower_id'),
```

3. Now we can expand the book editing form to include the new fields (**ch4 | Source Files | protected | views | book | _form.php**). Add a checkbox for the lendable field.

```
<div class="row">
  <?php echo $form->labelEx($model,'lendable'); ?>
  <?php echo $form->checkbox($model, 'lendable'); ?>
  <?php echo $form->error($model,'lendable'); ?>
</div>
```

Open a book form by either editing an existing book or creating a new one. Notice that the lendable field is, by default, checked. This is because, when we created the field in MySQL, we specified that by default the value will be `true`. We are very generous with lending our books.

126

4. We will add the borrower to the form as an Ajax autocomplete field providing a list of users. To support this, we must expand the Book model (**ch4** | **Source Files** | **protected** | **models** | **Book.php**) to include some transition fields. Add the following variables at the top of the Book class:

```
public $borrower_fullname = '';
public $borrower_fname;
public $borrower_lname;
```

Also add `borrower_fname` and `borrower_lname` to the searchable list in the rules function.

```
array('id, title, type_id, publication_date, value,
  price, notes, signed, grade_id, bagged, issue_number,
  lendable, borrower_fname, borrower_lname', 'safe',
  'on'=>'search'),
```

5. Now, we will add an Ajax function, named `aclist`, to the user controller (**ch4** | **Source Files** | **protected** | **controllers** | **UserController.php**) that will take the field input as a filter and return a list of users.

    i.   Add the new action to the admin access list in the User controller.

```
'actions'=>array('index', 'view', 'create', 'update',
  'delete', 'aclist'),
```

    ii.   Add the `Aclist` action to the User controller.

```
public function actionAclist($term)
{
  $results=array();
  $model = User::model();
  $criteria = new CDbCriteria();
  $criteria->with = array('person');
  $names = preg_split('/\W/', $_GET['term'], 2);

  if (count($names) == 1) {
    $criteria->addSearchCondition( 'person.fname',
      $names[0], true, 'OR');
    $criteria->addSearchCondition( 'person.lname',
      $names[0], true, 'OR');
  } else {
    $criteria->compare('person.fname', $names[0],
      true);
    $criteria->compare('person.lname', $names[1],
      true);
  }
  foreach($model->findAll($criteria) as $m)
```

```
      {
        $results[] = array(
          'id' => $m->{'id'},
          'label' => $m->person->{'fname'} .
          ' ' . $m->person->{'lname'},
          'value' => $m->person->{'fname'} .
          ' ' . $m->person->{'lname'},
        );
      }
      echo CJSON::encode($results);
    }
```

6. Create a function in the Book controller that joins the first and last name of a borrower into a single full name field.

```
public function set_fullname($model) {
  if ($model->borrower_id != null) {
    $model->borrower_fullname =
    ($model->borrower->person->fname ? $model-
      >borrower->person->fname . ' ' : '') .
    ($model->borrower->person->lname ? $model-
      >borrower->person->lname : '') ;
  }
}
```

7. Change the update action in the Book controller to use the `set_fullname` function, by first loading the model, then setting the full name value, and then calling the parent update function.

```
public function actionUpdate($id)
{
  $model=$this->loadModel($id);
  $this->set_fullname($model);
  $this->update($model);
}
```

8. In the Book Edit form (**ch4 | Source Files | protected | views | book | _form.php**), we add an auto-complete field for borrower. This field uses a Jui Zii extension, which uses jQuery to make the Ajax calls and process the responses.

```
<div class="row">
  <?php
    echo CHtml::activeHiddenField
      ($model,'borrower_id');
    echo $form->labelEx($model,'borrower');
    $this->widget('zii.widgets.jui.CJuiAutoComplete',
      array(
```

```
            'model' => $model,
            'attribute' => 'borrower_fullname',
            'sourceUrl' => array('user/aclist'),
            'name' => 'borrower_fullname',
            'options' => array (
              'minLength' => '3',
              'select'=> new CJavaScriptExpression('function(
                event, ui ) {
                $("#\'.CHtml::activeId
                  ($model,\'borrower_id\').\'")
                .val(ui.item.id);
                return true;
              }'
            ),
            'htmlOptions' => array (
              'size' => 32,
              'maxlength' => 32,
              'value' => $model->borrower_fullname,
            ),
      ));
    ?>
    </div>
```

Now, add a borrower, edit a book, and type three letters of the first or last name of a user. When you stop typing, you should see a drop-down list, containing the first name and last name entries for all of the matching users. If you are using the schema for the chapter, try `rie`. The results should be **Best Friend** and **Another Friend**.

9. Most of the time, a borrower will not be set when a book is created, so we must handle this case. (You can test it out by creating a book before taking this step.) Add an entry to the rules array in the Book model (**ch4 | Source Files | protected | models | Book.php**) to set the value of `borrower_id` to `null`, if no value is supplied.

    ```
    array('borrower_id', 'default', 'setOnEmpty' => true),
    ```

10. To see the new fields on the Book Index page, update the Book Index action in the Book controller (**ch4** | **Source Files** | **protected** | **controllers** | **BookController.php**) to create a `CDbCriteria` object with the `borrower.person` relation and include that object in the creation of the `CActiveDataProvider` object.

```
$criteria=new CDbCriteria;
$criteria->with = array('borrower.person');
$dataProvider=new CActiveDataProvider('Book', array(
  'criteria' => $criteria,
));
```

11. Add the following fields to **ch4** | **Source Files** | **protected** | **views** | **book** | **_view.php**:

```
<b><?php echo CHtml::encode($data-
  >getAttributeLabel('lendable')); ?>:</b>
<?php echo CHtml::encode($data->lendable); ?>
<br />

<?php
  echo "<b>" .
  CHtml::encode($data->getAttributeLabel('borrower')) .
  ":</b>";
  BookController::set_fullname($data);
  echo CHtml::encode($data->borrower_fullname);
?>
<br />
```

12. Then, update the Book View action to first load the model, then set the `fullname`, and pass it to the view as follows:

```
public function actionView($id)
{
  $model = $this->loadModel($id);
  $this->set_fullname($model);
  $this->render('view',array(
    'model'=>$model,
  ));
}
```

13. Add the fields to the attributes array in the book view (**ch4** | **Source Files** | **protected** | **views** | **book** | **view.php**), and the individual book page will display the new fields.

```
'lendable',
array(
  'name' => 'borrower',
  'value' =>
  $model->borrower_fullname,   ),
```

14. To display the new fields in the admin page, edit the Book model (**ch4** | **Source Files** | **protected** | **models** | **Book.php**), and add the following to the search function to search on the lendable field with the values `yes` or `no`:

```
$criteria->compare('lendable',
  ($this->lendable=="yes" ? 1:
($this->lendable=="no" ? 0 : "")),true);
```

Add these to include the related borrower and person information and to search on the borrower's first and last names.

```
$criteria->compare('person.fname', $this-
  >borrower_fname, true);
$criteria->compare('person.lname', $this-
  >borrower_lname, true);
$criteria->with = array('borrower.person');
```

15. Also, add a `CSort` object to the search function as follows:

```
$sort = new CSort;
$sort->attributes = array(
  'borrower_fname' => array(
    'asc' => 'person.fname',
    'desc' => 'person.fname DESC',
  ),
  'borrower_lname' => array(
    'asc' => 'person.lname',
    'desc' => 'person.lname DESC',
  ),
  '*',
);
```

16. Pass the sort variable to the `CActiveDataProvider` object as follows:

```
return new CActiveDataProvider($this, array(
  'criteria'=>$criteria,
  'sort'=>$sort,
));
```

17. Finally, add the columns to the admin view (**ch4** | **Source Files** | **protected** | **views** | **book** | **admin.php**) list.

```
array(
  'name' => 'lendable',
  'header' => 'Lendable',
  'value' => '(($data->lendable == 1) ? "yes" : "no")',
),
array(
```

```
          'name' => 'borrower_fname',
          'header' => 'Borrower First Name',
          'value' => '(($data->borrower != null) ?
            $data->borrower->person->fname . \' \' : \'\')',
      ),
      array(
          'name' => 'borrower_lname',
          'header' => 'Borrower Last Name',
          'value' => '(($data->borrower != null) ?
            $data->borrower->person->lname . \' \' : \'\')',
      ),
```

The final Book admin screen will look like the following screenshot:

Title	Issue Number	Type	Publication Date	Value	Price	Lendable	Borrower First Name	Borrower Last Name	
Batman		2	2012-05-08	50.00	3.00	yes	Comic	Fan	
The Amazing Spider Man		1	2012-02-13	10.00	3.00	yes	Best	Friend	
X-Men		3	0000-00-00	5.00	1.00	yes	Another	Friend	
Sandman		1	2012-05-17	8.00	3.00	no			

Objective Complete - Mini Debriefing

To add the library management piece, we started by adding fields to the book table in the database. We updated the model to include the new fields. Then piece by piece, we added support for the new field in each of the book views. One area that we did not touch on was the **Advanced Search** form. If you would like, you can update this form to include the fields that you find useful for searching, but do not necessarily need for quick searching. To do this, edit **ch4** | **Source Files** | **protected** | **views** | **book** | **_search.php**.

Adding User Functions – Library

Because we like to share books from our collection with our friends, we will add a lending function to our site. When a friend logs in, he/she will be able to see which books are available for borrowing, which he/she is currently borrowing, and which he/she has borrowed in the past.

Prepare for Lift Off

To demonstrate the effect of lendable/not lendable in the library, edit a few books and turn off the **Lendable** field.

Engage Thrusters

1. To access the new feature, we need to add a **Library** option to the menu, in the file **ch4 | Source Files | protected | views | layouts | main.php**.

    ```
    array('label'=>'Library', 'url'=>array('/library/index')),
    ```

    Log in to the site and you will see the updated menu, but clicking on this link will cause a *page not found (404)* error. We must create a new controller and view to support it.
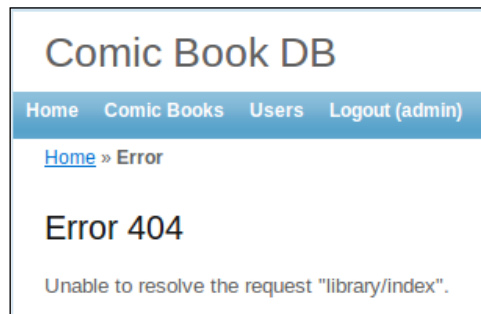
2. Right-click on the **controllers** folder (**ch4** | **Source Files** | **protected** | **controllers**) and select **New** | **PHP File**.
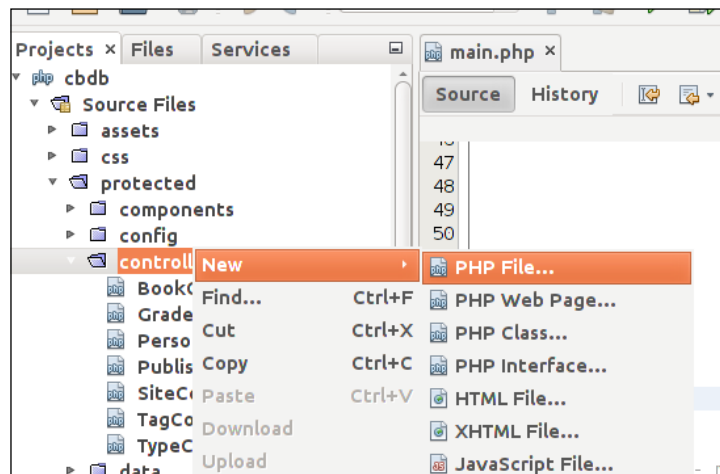
3. Enter `LibraryController` in the **File Name** field and click on **Finish**.
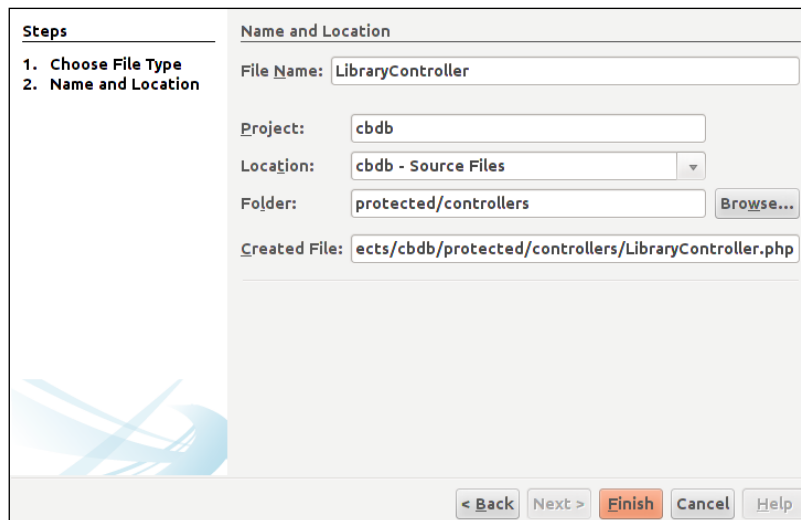
4. Paste the following contents into the file:

```php
<?php

class LibraryController extends Controller
{
```

```php
    /**
     * @return array action filters
     */
    public function filters()
    {
      return array(
        'accessControl',
          // perform access control for CRUD operations
      );
    }

    /**
     * Specifies the access control rules.
     * This method is used by the 'accessControl' filter.
     * @return array access control rules
     */
    public function accessRules()
    {
      return array(
        array('allow',
          // allow all users to perform 'index' action
          'actions'=>array('index'),
          'users'=>array('*'),
        ),
        array('deny',  // deny all users
          'users'=>array('*'),
        ),
      );
    }

    /**
     * Display library
     */
     public function actionIndex()
     {
         $criteria=new CDbCriteria;
         $criteria->compare('lendable', 1);
         $dataProvider=new CActiveDataProvider
           ('Book', array(
              'criteria' => $criteria,
         ));
         $this->render('index',array(
             'dataProvider'=>$dataProvider,
         ));
     }
}
```

> We have chosen to explicitly state that access rules apply to all users. You can leave the user line out, if you prefer, as it is the default value.

We have added a controller with access control enabled, default layout, and one action, that is Index, which returns all the books that are lendable and not currently lent. Now if you click on **Library** in the menu, the error will look even worse!


```
CException

LibraryController cannot find the requested view "index".

/opt/lampp/htdocs/yii-1.1.10.r3566/framework/web/CController.php(879)

867          {
868              if(($viewFile=$this->getViewFile($view))!==false)
869              {
870                  $output=$this->renderFile($viewFile,$data,true);
871                  if($processOutput)
872                      $output=$this->processOutput($output);
873                  if($return)
874                      return $output;
875                  else
876                      echo $output;
877              }
```

5. We will fix the error by creating a view to present the results of the Library controller Index action. Begin by creating a new view folder. In NetBeans, right-click on **views** under **ch4** | **Source Files** | **protected** and select **New** | **Folder**.

6. Enter `library` in the **Folder Name**, and click on **Finish**.

7. Right-click on the new library folder and select **New** | **PHP File**. Name the file `index` and click on **Finish**.

8. Replace the contents of the new `index.php` file with the following grid view:

```php
<?php $this->widget('zii.widgets.grid.CGridView', array(
  'id'=>'book-grid',
  'dataProvider'=>$dataProvider,
  'columns'=>array(
    'title',
    'issue_number',
   array(
     'name' => 'Type',
     'header' => 'Type',
     'value' => '$data->type->name',
```

```
    ),
    array(
      'name' => 'Publisher',
      'header' => 'Publisher',
      'value' => '(($data->publisher!=null) ?
        $data->publisher->name : \'\')',
    ),
    array(
      'name' => 'Authors',
      'header' => 'Authors',
      'value' => array
        ($dataProvider->model, 'author_list'),
    ),
    'publication_date',
    array (
      'class'=>'CButtonColumn',
      'template'=>'{request}',
      'buttons' => array(
        'request' => array(
          'label' => 'Request',
          'imageUrl' => Yii::app()->baseUrl .
            '/images/request_lozenge.png',
          'url' => 'Yii::app()->createUrl
            ("library/request", array("id"=>$data->id))',
        ),
      ),
    ),
  ),
)); ?>
```

9. In the library index, the **Authors** column takes advantage of a `CGridView` feature that allows you to specify an object and a function to supply the value for the column. In this case, we call the `author_list` function in the Book model. Open up the Book model and add that function.

```
public function author_list($data, $row) {
  $list = array();
  $count = 1;
  foreach ($data->authors as $a) {
    $list[] = $a->fname . ' ' . $a->lname;
    $count++;
  }
  return implode(",", $list);
}
```

Next, we will add a Book model function to display the book's lending status in the status column.

Title	Issue Number	Type	Publisher	Authors	Publication Date	
Green Lantern		issue	Pub Co		2012-05-01	Request
Witchblade		issue	Pub Co		2000-03-01	Request
300		graphic novel			2002-10-01	Request
Wolverine		issue			1982-05-01	Request
Hellboy	4	trade	Pub Co	Comic Writer	2012-05-01	Request
Test Book	3	trade			2012-06-15	Request

10. It will be nice to see the lending status of the books, whether we have them or someone else does. Let's add a status column to the library index view.

```
'publication_date',
array(
  'name' => 'Status',
  'header' => 'Status',
  'value' => array($dataProvider->model, 'get_status'),
),
array (
  'class'=>'CButtonColumn',
```

11. Also, add a supporting function to the Book model.

```
public function get_status($data, $row) {
  $status ="";
  if ($data->borrower_id != null) {
    $status = "Checked Out";
  }
  if ($data->borrower_id ==
    Yii::app()->user->getId()) {
    $status = "You Have It";
  }
  if ($status == null) {
    $status = "Available";
  }
  return $status;
}
```

12. Also in the Library index view, we added a custom link to request a book. We must add the action to the Library controller (**ch4 | Source Files | protected | controllers | LibraryController.php**) to process the request. But first, we will need a table to record the request.

```
CREATE TABLE 'request' (
  'book_id' int(10) unsigned NOT NULL,
  'requester_id' int(10) unsigned NOT NULL,
  PRIMARY KEY ('book_id','requester_id'),
  KEY 'requester_id' ('requester_id'),
  KEY 'book_id' ('book_id'),
  CONSTRAINT  FOREIGN KEY ('book_id') REFERENCES
    'book' ('id'),
  CONSTRAINT FOREIGN KEY ('requester_id') REFERENCES
    'user' ('id')
) ENGINE=InnoDB DEFAULT
```

13. Use Gii to generate a model from the request table.

14. Add the new relation to the Book model (**ch4 | Source Files | protected | models | Book.php**).

```
'requesters' => array(self::MANY_MANY, 'User',
  'request(requester_id, book_id)',
'index'=>'id'),
'requests' => array(self::HAS_MANY, 'Request',
'book_id', 'index' => 'requester_id'),
```

15. Create the request action in the Library controller (**ch4 | Source Files | protected | controllers | LibraryController.php**).

```
public function actionRequest($id)
{
  $request = new Request();
  $request->book_id = $id;
  $request->requester_id = Yii::app()->user->getId();
  $request->save();
  Yii::app()->user->setFlash ('success', "Your book
    request has been submitted.");
  $this->redirect(array('index'));
}
```

16. Add the access authorization to the Library controller access rules.

```
'actions'=>array('index', 'request'),
```

17. Add a flash display at the top of the Library index view to display success when a request is recorded. A flash message keeps the message in session through one or more of the user's requests.

```
<?php if(Yii::app()->user->hasFlash('success')) { ?>
  <div class="flash-success">
    <?php echo Yii::app()->user->getFlash('success'); ?>
  </div>
<?php } ?>
```

18. Add the parameter **visible** to the `CButtonColumn` in the Library index view to display the Request link if no request has been made. Once again, we will use a function defined on our model to get our result.

```
'request' => array(
  'label' => 'Request',
  'imageUrl' => Yii::app()->baseUrl .
    '/images/request_lozenge.png',
  'url' => 'Yii::app()->createUrl("library/request",
    array("id"=>$data->id))',
  'visible' => array($dataProvider->model, 'requested'),
),
```

19. Add the function `requested` to the Book model to support the change we just made to the view. The function will return true or false to toggle the **Request** link in the library grid.

```
public function requested($row, $data) {
  $me = Yii::app()->user->getId();

  foreach ($data->requesters as $r) {
    if ($r->id == $me) {
      return false;
    }
  }
  if ($data->borrower_id==$me) {
    return false;
  }
  return true;
}
```

The finished Library view looks like the following screenshot:

Title	Issue Number	Type	Publisher	Authors	Publication Date	Status	
Batman		graphic novel		Bob Miller, Test Author2	2012-05-08	You Have It	
The Amazing Spider Man		trade			2012-02-13	Checked Out	
X-Men		issue			0000-00-00	Checked Out	
Green Lantern		issue	Pub Co		2012-05-01	Available	Request
Witchblade		issue	Pub Co		2000-03-01	Available	Request
300		graphic novel			2002-10-01	Available	
Wolverine		issue			1982-05-01	Available	Request
Stardust		trade	Pub Co		0000-00-00	You Have It	
Hellboy	3	trade	Pub Co	Mike Comic, Lauren Person	2012-05-24	Checked Out	Request

20. Now we will circle back to the book list view to display the requests and add the functions to process them. To display the list of requests for each book in the index view, add this section to the bottom of **ch4** | **Source Files** | **protected** | **views** | **book** | **_view.php**.

```php
<?php
  if ($data->requesters) {
    echo "<strong>Requests</strong>\n";
    echo "<ul>\n";
    foreach ($data->requesters as $r) {
      echo "<li>" . CHtml::encode($r->person->fname .
        ' ' . $r->person->lname) .
      " " . CHtml::link("Lend",
        array("library/lend",
          "book_id" => $data->id,
      "user_id" => $r->id)) .
      "</li>";
    }
    echo "</ul>\n";
  }
?>
```

21. To add a link next to a borrower to process a book return, change the entry for borrower in the book view to include a return link.

```php
<?php
echo "<b>" .
CHtml::encode($data->getAttributeLabel('borrower')) .
":</b>";
BookController::set_fullname($data);
echo CHtml::encode($data->borrower_fullname) . " "
  . CHtml::link("Return", array("library/return",
  "book_id" => $data->id, "user_id" =>
  $data->borrower_id));
?>
<br />
```

22. Also, update the Library controller with this function to process a request.

```php
public function actionLend($book_id, $user_id)
{
  $model=Book::model()->findByPk($book_id);
  if($model===null)
    throw new CHttpException(404,'The requested book
      does not exist.');
  $request = Request::model()->find(
    'book_id=:book_id AND requester_id=:user_id',
      array(
      ':book_id' => $book_id,
      ':user_id' => $user_id,
  ));
  if($request===null)
  throw new CHttpException(404,'The request does not
    exist.');

  $request->delete();
  $model->borrower_id = $user_id;
  $model->save();
  $this->redirect(array('book/index'));
}
```

23. The twin function to `lend` is `return`:

```php
public function actionReturn($book_id,$user_id)

  {
  $model=Book::model()->findByPk($book_id);
  if($model===null)
    throw new CHttpException(404,'The requested book
      does not exist.');
  $model->borrower_id = null;
```

```
        $model->save();
        $this->redirect(array('book/index'));
    }
```

24. Add the new action to the access rules. For the moment, it is ok to permit it for all users. We will adjust the permissions in the next task.

```
array('allow',
    // allow all users to perform 'index' action
    'actions'=>array('index', 'request', 'lend', return),
    'users'=>array('*'),
),
```

25. The resulting Book index page will now include a list of requests, if any requests for the book are pending.

ID: 1
Title: Batman
Issue Number:
Type: 2
Publication Date: 2012-05-08
Value: 50.00
Price: 3.00
Notes:
Lendable: 1
Borrower: Comic Fan

ID: 2
Title: The Amazing Spider Man
Issue Number:
Type: 1
Publication Date: 2012-02-13
Value: 10.00
Price: 3.00
Notes: This book is awesome.
Lendable: 1
Borrower: Best Friend
Requests
  • Comic Fan Lend

Objective Complete - Mini Debriefing

We have added a new action to the site that is related to an existing model, but uses its own controller. We used a function call from `CGridView` to display more complex column information, we used `CButtonColumn` to define a new button action, **Request**, and we added a condition to hide the button if a request has already been created. For the management piece, we updated the comic book index page to display any pending requests. We included a convenience link, **Lend,** that quickly updates the borrower, so that we don't have to navigate to the comic book update page, search for the new borrower, and save the changed record.

Defining Roles and Access

This task is mostly about planning. When you add RBAC to your application, you will need to decide how the access to your system will be allocated. A good place to start is to look at the actions in your current system and the roles you think you will need.

Yii defines RBAC in terms of roles, tasks, and operations. An operation is a single action on an object. We will set its name as the object followed by the action. For example, the name for the operation consisting of the action `Create` on the object `Book` would be `bookCreate`. A task is a named collection of operations. For example, you might collect all of the user management operations (`userCreate`, `userDelete`, `userUpdate`, and `userView`) into a single task named `manageUser`. A role is a collection of tasks and operations and other roles. Assigning one role to another creates a hierarchy and is a convenient way for the user to manage nested levels of access. For example, if you have the roles Reader, Contributor, and Administrator, it makes sense that a contributor will have all of the permissions of a reader, and an administrator will have all of the permissions of a contributor.

Administrator > Contributor > Reader

A user is authorized to perform any of the actions collected under a role that is assigned to him/her.

Engage Thrusters

1. For this project, we will define roles as follows, in order of decreasing levels of access:

    i. Authority – your role, with total access

    ii. Administrator – can add and edit book and user entries

    iii. Borrower – can view the library list and make requests

    iv. Viewer – can view the comic book collection

    v. wishlistAccess – can only view the wishlist

2. Expand this list of roles in terms of the actions they can perform:

    i. Authority: Create/read/update/delete role, task, operation

    ii. Administrator: Create/read/update/delete user, book, wish

    iii. Borrower: Read book, wish, library, and make library request

    iv. Viewer: Read book and wish

    v. WishlistAccess: Read wish and read/update own user entry

3. Yii provides a function for scripting and loading all of the roles, tasks, and operations we just defined. First, we will have to set up the database to hold these new entities, and it is pretty simple. All you need are three tables to capture three things:

   ❑ Authorization items

   ❑ Authorization hierarchy

   ❑ Assignment

4. Go to the **Servers** tab, and use the **Execute Command** tool to create the following tables (the table definitions can be found in the directory `protected/data/auth_tables.sql`):

```
CREATE TABLE 'auth_item' (
   'name'                 varchar(64) NOT NULL,
   'type'                 int NOT NULL,
   'description'          text,
   'bizrule'              text,
   'data'                 text,
   PRIMARY KEY ('name')
) ENGINE=InnoDB

CREATE TABLE 'auth_item_child' (
   'parent'        varchar(64) NOT NULL,
   'child'         varchar(64) NOT NULL,
   PRIMARY KEY ('parent', 'child'),
   FOREIGN KEY ('parent') REFERENCES 'auth_item' ('name')
     ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY ('child') REFERENCES 'auth_item' ('name')
     ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB

CREATE TABLE 'auth_assignment' (
   'itemname'      varchar(64) NOT NULL,
   -- NOTE - userid is the format the yii libraries expect
   'userid'        int(10) unsigned,
   'bizrule'       text,
   'data'          text,
   PRIMARY KEY ('itemname', 'userid'),
   FOREIGN KEY ('itemname') REFERENCES 'auth_item'
     ('name') ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB
```

In the next section, we will download and install a Yii extension that will provide a nice graphical interface for RBAC management. Before we do that, we will initialize the RBAC with a script built from the hierarchy we defined in the previous steps. We have included the full script in the chapter files (**ch4 | Source Files | protected | command | shell | RbacCommand.php**). We will touch on some key excerpts next.

For many of our controllers, we have already created more actions than the Gii-generated CRUD. For example, there are nine operations on the `Wish` object:

```
$auth->createOperation('WishAdmin','admin access to
  wishes');
$auth->createOperation('WishIndex','index of wishes');
$auth->createOperation('WishCreate','create a wish');
$auth->createOperation('WishView','read a wish');
$auth->createOperation('WishUpdate','update a wish');
$auth->createOperation('WishDelete','delete a wish');
$auth->createOperation('WishClaim','claim a wish');
$auth->createOperation('WishRemoveAuthor','remove an author
  from a wish');
$auth->createOperation('WishCreateAuthor','create an author
  for a wish');
```

You may notice that the format of the operation name is `<Controller><Action>`. The reason for this is that the management module that we will add later relies on this naming convention.

Each role in our hierarchy may be assigned the previous role and any new operations that belong to it.

```
$role=$auth->createRole('borrower');
$role->addChild('viewer');
$role->addChild('LibraryIndex');
$role->addChild('LibraryRequest');
```

5. Update the app configuration file (**ch4 | protected | config | main.php**) to activate the included Authorization Manager component.

```
'components'=>array(
  'user'=>array(
    // enable cookie-based authentication
    'allowAutoLogin'=>true,
  ),
  'authManager' => array(
    'class' => 'CDbAuthManager',
    'connectionID' => 'db',
    'assignmentTable' => 'auth_assignment',
    'itemTable' => 'auth_item',
    'itemChildTable' => 'auth_item_child',
```
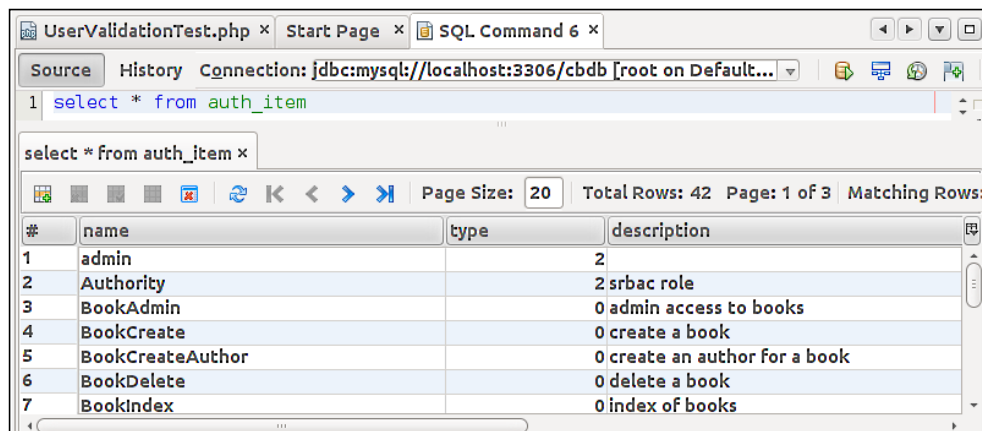
```
        ),
        'urlManager'=>array(
```

6. Run the script from the command line with the Yii shell tool to load the hierarchy into the database.

```
cd ~/project/ch4

yiic shell

>> rbac
```

The command tool will output **Rbac initialized.** You can exit the tool with the command **exit**.

7. You can verify the initialization by going to the NetBeans server view, connecting to your database, right-clicking on any of the authorization tables, and selecting the **View Data** command.

Objective Complete - Mini Debriefing

When you implement a role-based access control system, you will need to give some thought to the roles that are needed in your system, the actions each role may perform, and the relationship between the roles. A review of the actions currently in your system can help you generate a list of roles. A diagram of roles and actions can help you spot gaps in your coverage.

Classified Intel

At this point, you could configure your project and move from the file-based authorizations in each controller to the database authorizations that we have initialized, but maintaining and building on this information will be difficult without a more intuitive interface. The Yii team's stated intention was for developers to make suitable RBAC interfaces for their projects. You can create your own, but there are some extensions available that do a good job of meeting most RBAC management needs. So we will install one of those extensions to demonstrate a helpful graphical management interface and then complete the configuration and activate the new authorization system.

Adding the RBAC Extension

In this task, we continue to configure and activate the role-based access control system, and add a management interface to the website to make it easier to review, add to, update, and maintain the authorization information.

Prepare for Lift Off

Download and unpack the `srbac` and `YiiSmartMenu` extensions from the Yii website:

- ▶ `http://www.yiiframework.com/extension/srbac/`
- ▶ `http://www.yiiframework.com/extension/yiismartmenu/`

Engage Thrusters

1. Create a directory named `modules` in your project's `protected` directory.

   ```
   cd ~/projects/ch4/protected
   mkdir modules
   ```

2. Move the `srbac` directory into the newly created `modules` directory.

   ```
   mv ~/Downloads/srbac\ 1.3beta/srbac/
     ~/project/ch4/protected/modules/.
   ```

3. Edit the project configuration file (**ch4** | **protected** | **config** | **main.php**) to import the `srbac` module.

   ```
   'import'=>array(
     'application.models.*',
     'application.components.*',
     'application.modules.srbac.controllers.SBaseController',
   ),
   ```

4. In the same config file, change the `AuthManager` component to use the `srbac` module.

```
'authManager' => array(
  'class' => 'application.modules.srbac.
   components.SDbAuthManager',
  'connectionID' => 'db',
```

5. Also in the `config` file, add an entry for `srbac` to the `modules` array. This entry will configure the behavior of the module.

```
'srbac' => array(
  'userclass'=>'User', //default: User
  'userid'=>'id', //default: userid
  'username'=>'username', //default:username
  'delimeter'=>'@', //default:-
  'debug'=>true, //default :false
  'pageSize'=>10, // default : 15
  'superUser' =>'Authority', //default: Authorizer
  'css'=>'srbac.css', //default: srbac.css
  'layout'=>
  'application.views.layouts.main',
  'notAuthorizedView'=>
    'srbac.views.authitem.unauthorized',
  'alwaysAllowed'=>array( 'SiteLogin',
    'SiteLogout','SiteIndex', 'SiteError'),
  'userActions'=>array('Show','View','List'),
  'listBoxNumberOfLines' => 15, //default : 10
  'imagesPath' => 'srbac.images',
    // default: srbac.images
  'imagesPack'=>'noia', //default: noia
  'iconText'=>true, // default : false
  'header'=>'srbac.views.authitem.header',
  'footer'=>'srbac.views.authitem.footer',
  'showHeader'=>true, // default: false
  'showFooter'=>true, // default: false
  'alwaysAllowedPath'=>'srbac.components',
),
```

The following points should be noted in this configuration:

❑ For the first run, we leave debugging on, which allows every user access to all screens. We leave the access to srbac open until we have used the interface to configure our admin user to have the authority permission.

❑ The **alwaysAllowed** line lists actions that are public on your site. Put items in this list that you want to be always accessible and do not wish to configure further.

149

6. Create a file (**ch4** | **Source Files** | **protected** | **modules** | **srbac** | **components** | **allowed.php**) that contains an empty array.

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

Set the permissions on this file to permit writing from the web server.

chown lomeara:www-data allowed.php

You will be able to configure the contents of this file via the srbac web interface.

7. Remove the `install` folder from the project (**ch4** | **Source Files** | **protected** | **modules** | **srbac** | **views** | **authitem** | **install**), because it provides access to administrative functions that are no longer needed.

8. Now access the srbac page `http://localhost/cbdb/index.php/srbac`.

9. Click on the **Managing auth items** button. The screen will update with a list of the Roles, Operations, and Tasks that were created by our script.
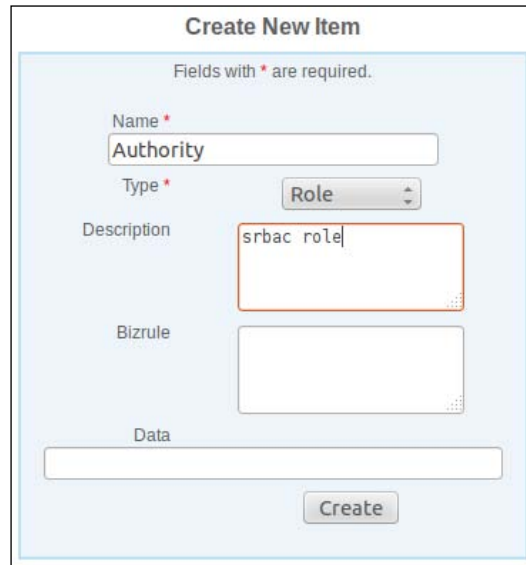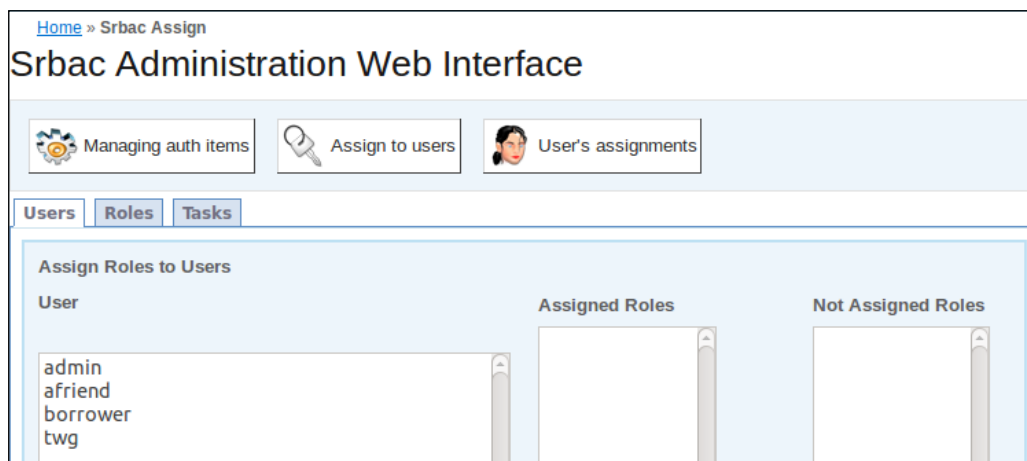
10. Click on the **Create** button next to the **Search** bar and in the form that appears, set **Name** to **Authority**, **Type** to **Role**, and **Description** to **srbac role**.

11. Click on the **Create** button in the form to save the new role.

12. Click on the **Assign to users** button.

13. Select the user(s) who will have access to srbac from the left-hand column. For our examples, select the username `admin`. Select **Authority** in the right-hand column, and click on the left arrow button [<<] to move from **Not Assigned Roles** to **Assigned Roles**. As a result, the user, `admin`, will have the roles **admin** and **Authority** in the **Assigned Roles** column.
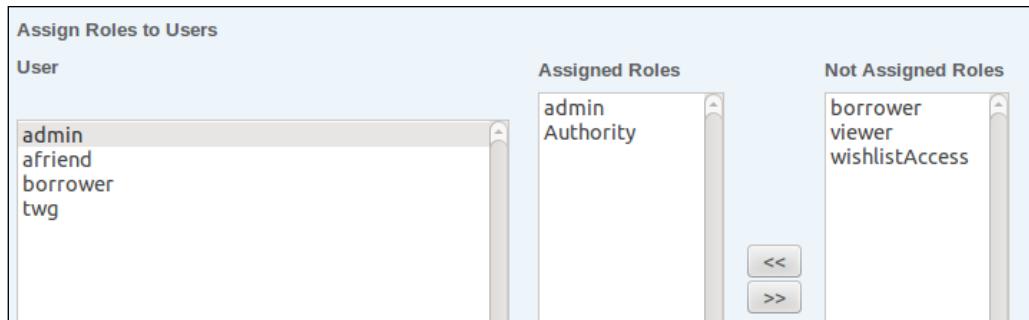
14. Now you can go back into the `config` file (**ch4** | **Source Files** | **protected** | **config** | **main.php**) and comment out the srbac debug line, so that only the admin user can access srbac.

15. For our own convenience, we will add an srbac link to the site menu. Add the following line to **ch4** | **Source Files** | **protected** | **views** | **layouts** | **main.php**:

```
array('label'=>'Srbac', 'url'=>array('/srbac'),
   'authItemName' => 'Authority'),
```

For this menu item, we assign the authorized role value, `Authority`, to `authItemName` to check the access instead of depending on the URL, because srbac permits access based on this role and does not, as we have configured it, have the operations defined to match the URLs.

16. To display the applicable contents of the Comic Book menu to all users, we apply this method to the comic book item in the site menu in **ch4** | **Source Files** | **protected** | **views** | **layouts** | **main.php**. Our lowest level access user, and all users up the hierarchy, should be able to see some items in the Comic Book menu, so we include the lowest level role as the value for `authItemName`.

```
array(
   'label'=>'Comic Books',
   'url'=>array('/book/index'),
   'items' => array(
     array('label'=>'Publishers',
     'url'=>array('/publisher/index')),
```

```
      array('label'=>'WishList',
      'url'=>array('/wish/index')),
      array('label'=>'Library',
      'url'=>array('/library/index')),
    ),
    'authItemName' => 'WishlistAccess',
  ),
```

The Comic Book link will be visible and active, but if a user without authorization selects it, he will receive an error message.
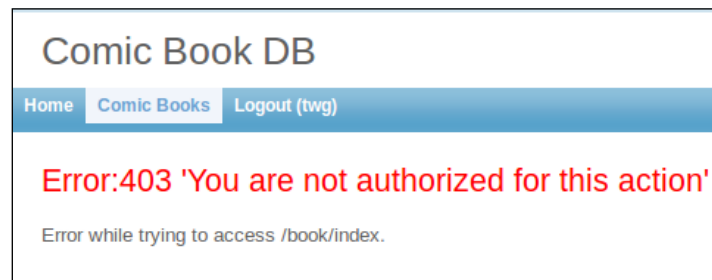
17. The site currently uses a component, `AuthMenu`, that displays menu items based on authorization. We built this component in a previous chapter based on a distributed component named `YiiSmartMenu` to use the file-based authorization configuration. Now we will replace that component with `YiiSmartMenu`. Copy the file from the extension directory into your project.

    **cp ~/Downloads/<Yii Smart Menu directory>YiiSmartMenu.php
      ~/projects/ch4/protected/components/.**

18. Change the value of the member variable `partItemSeparator` in the `YiiSmartMenu` class from "." to "" in **ch4 | Source Files | protected | components | YiiSmartMenu.php**.

    ```
      public $partItemSeparator = "";
    ```

19. Remove the old `AuthMenu` file (**ch4 | Source Files | protected | components | AuthMenu.php**) from the project.

20. Replace the occurrences of `AuthMenu` in **ch4 | Source Files | protected | views | layouts | column2.php** and **ch4 | Source Files | protected | views | layouts | main. php** with `YiiSmartMenu`.

21. Activate the srbac by changing the parent class of **ch4 | Source Files | protected | components | Controller.php** to `SBaseController`.

    ```
    class Controller extends SBaseController
    ```

22. Remove or comment out the `accessControl` filter in every controller:

   ❑ `BookController`

   ❑ `LibraryController`

   ❑ `PublisherController`

   ❑ `UserController`

   ❑ `WishController`

```
public function filters()
{
  return array(
    //'accessControl',
    // perform access control for CRUD operations
  );
}
```

23. Remove or comment out the `accessRules` function in every controller, because it is no longer needed.

24. Try out access control by logging in as a user with fewer privileges. See the *Classified Intel* section for more information about test users that are already in the database.

Objective Complete - Mini Debriefing

In this section, we explained how to install, configure, and activate the srbac extension. The extension comes with its own set of instructions, but we wanted to offer a more detailed explanation to help you set it up. Also, our approach to the database tables, naming of roles, and organization of the authorization hierarchy is a little different. The configuration example we provided is slightly different from the default.

Classified Intel

You may have noticed in the initial RBAC configuration or when browsing the srbac interface that some users have been assigned to the various roles. We supplied these users in the schema for the chapter with the password for each set to `test`. We configured each one to a different role, so that you can test user experience for each role. The full list of users to roles is:

▸ `admin => admin`

▸ `borrower => borrower`

▸ `afriend => viewer`

▸ `twg => wishlistAccess` (`twg` is short for "the wish giver")

Adding Roles to User Management

As you can see, the interface provided by the srbac extension is very powerful and convenient. It is great for adding new roles, tasks, and operations. However, when we add new users to the system, it would be convenient to assign the users, roles on the spot.

Engage Thrusters

1. First, add the relationship to the assignment object in the User model (**ch4** | **Source Files** | **protected** | **models** | **User.php**).

    i.   Add a comment at the top of the file listing the new model variable in the model relations section as follows:

    ```
    * The following are the available model relations:
     * @property Person $person
     * @property Assignments[] $assignments
    ```

    ii. Add an entry for assignments to the `relations` array as follows:

    ```
    return array(
       'person' => array(self::BELONGS_TO, 'Person',
         'person_id'),
       'assignments' => array(self::HAS_MANY, 'Assignments',
         'userid'),
    );
    ```

2. Display the assignments in the User form (**ch4** | **Source Files** | **protected** | **views** | **user** | **_form.php**):

    ```
    <div class="row">
      <ul>
        <?php foreach($user->assignments as $a) {
          echo "<li>" . $a->itemname . "</li>";
        } ?>
      </ul>
    </div>
    ```

The currently assigned roles will now appear in the User edit screen as shown in the following screenshot:

3. Now we need to display the roles that have not been assigned to this user, and provide a means to add an assignment.

   Change the list display to use a `renderPartial` function as follows:

   ```
   <div class="row">
     <b>Assignments</b><br/>
     <ul class="roles">
       <?php foreach($user->assignments as $a) {
         echo $this->renderPartial('//includes/role_li',
           array(
             'user' => $user,
             'assignment' => $a,
         ));
       } ?>
     </ul>
   ```

4. Create a new view file for the `renderPartial` function call in the directory **ch4 | Source Files | protected | views | includes | role_li.php** with the following contents:

   ```
   <?php
     echo "<li id=\"role-" . $assignment->itemname. "\">" .
       $assignment->itemname .
   ```

```
       " <input class=\"revoke\" type=\"button\" " .
         "onClick=\"revoke('" .
           Yii::app()->controller-
             >createUrl("/user/revokeRole",
             array("id" => $user->id,
               "role_name"=>$assignment->itemname,
           "ajax"=>1)) . "', '" .
         $assignment->itemname . "')\" " .
         "value=\"Revoke\" " .
       "/>" .
     "</li>";
```

5. Add a section to display a picklist of unassigned roles:

```
<b>Un Assigned Roles</b><br/>
<?php echo SHtml::activeDropDownList($user,'role',
  SHtml::listData(
    $user->getUnassignedRoles(), 'name', 'name'),
array('size'=>5, 'class'=>'dropdown')); ?>
<br/>
<input class="add" type="button"
  obj="User"
  url="<?php
    echo Yii::app()->controller->createUrl(
      "/user/assignRole",
  array("id"=>$user->id)); ?>"
value="Add"/>
```

6. Create a new class variable in the User model to hold the role value as follows:

```
public $person_fname;
public $person_lname;
public $role;
```

7. Add a function that wraps the srbac helper function `getUserNotAssignedRoles` to return the list of roles that have not been assigned to this user.

```
public function getUnassignedRoles()
{
  return(Helper::getUserNotAssignedRoles($this->id));
}
```

8. Now on the User edit screen, we can choose from a list of unassigned roles, to assign to the user.

9. Create the assign role action in the User controller.

```
public function actionAssignRole($id)
{
  // request must be made via ajax
  if(isset($_GET['ajax']) && isset($_GET['role'])) {
    $model=$this->loadModel($id);
    $auth = Yii::app()->authManager;
    $role =CHttpRequest->getParam('role');
    $auth->assign($role, $id, '', '');
    $role=Assignments::model()->find("itemname='" .
      $role . "'");
    $this->renderPartial('//includes/role_li',array(
      'user'=>$model,
      'assignment'=>$role,
    ), false, true);
  }
  else
  throw new CHttpException(400,'Invalid request.');
}
```

10. Create the revoke role action in the User controller.

```
public function actionRevokeRole($id)
{
```

```
      // request must be made via ajax
      if(isset($_GET['ajax'])) {
        $auth = Yii::app()->authManager;
        $auth->revoke($_GET['role_name'], $id);
      }
      else
      throw new CHttpException(400,'Invalid request.');
    }
```

11. Create a JavaScript file containing these functions to support the assign and revoke buttons.

```
function revoke(url, role_name) {
  var role = "#role-" + role_name;
  $.ajax({
    type: 'get',
    url: url,
    success: function(resp) {
      $('li').remove(role);
    },
    error: function() {
      alert('error!');
    }
  });
}

$(document).ready(function() {
  $('.assign').click(function()$
    ('.assign').click(function() {
    // initialize data object - it's ajax
    data = {
      "role" : $("#User_role").val(),
      "ajax" : "1"
    };
    $.ajax({
      type: 'get',
      url: $(this).attr('url'),
      data: data,
      success: function(resp) {
        $("ul.roles").append(resp);
      },
      error: function() {
        alert('error!');
      }
    });
  });
});
```

12. Add `clientScript` calls to the beginning of `update.php` and `create.php` in the User view directory (**ch4** | **Source Files** | **protected** | **views** | **user**) to include jQuery and the new custom script.

```
Yii::app()->getClientScript()->registerCoreScript
  ( 'jquery.ui' );
Yii::app()->clientScript->registerScriptFile(
  Yii::app()->request->baseUrl . '/js/user_form_ajax.js'
);
```

13. Go to the srbac screen to add these new actions and assign them to the `admin` role, so that access to them is limited. On the srbac main screen, click on the **Managing auth items** button, and then click on the **Create** button.
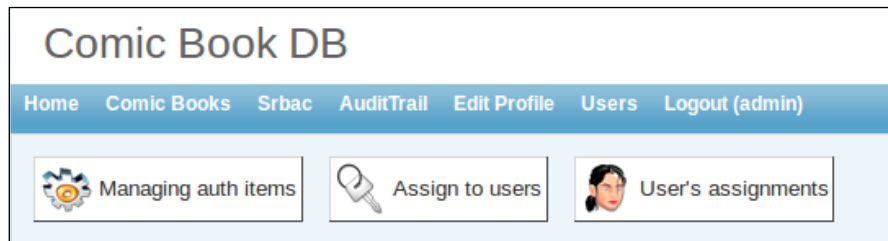
14. Enter `UserAssignRole` as the name. Leave **Type** as **Operation**. Enter a description, if you like. Click on the **Create** button in the **Create New Item** form when you are done.

15. Click on the **Create** button on the left-hand side of the screen to commence adding a new operation.

16. Enter `UserRevokeRole` as the name. Leave **Operation** as the type. Enter a description, if you like. Click on **Create** at the bottom of the form when you are done.

17. Click on the **Assign to Users** button at the top of the page.

18. Go to the **Tasks** tab, and select **manageUser** under the **Task** column.

19. Select **UserAssignRole** and **UserRevokeRole** from the **Not Assigned Operations** column and click on the left arrow button [`<<`] to move **Operations** to the **Assigned Operations** column.

20. The actions to assign and revoke work. They update the list of assigned roles, but they do not update the select list of unassigned roles.

21. To add this feature, start by moving the select list out of the `_form.php` view and into its own view in **ch4 | Source Files | protected | views | includes | role_select.php**.

```
<div id="role_list">
  <b>Un Assigned Roles</b><br/>
  <?php echo CHtml::activeDropDownList($user,'role',
    SHtml::listData(
      $user->getUnassignedRoles(), 'name', 'name'),
  array('size'=>5, 'class'=>'dropdown')); ?>
  <br/>
  <input class="assign" type="button"
    onClick="assign('<?php
      echo Yii::app()->controller->createUrl(
        "/user/assignRole",
      array("id"=>$user->id)); ?>','<?php
        echo Yii::app()->controller->createUrl(
          "/user/reloadRoles",
      array("id"=>$user->id)); ?>')"
  value="Add"/>
</div>
```

We wrap it in a div tag with the ID `role_list` so we can easily replace it when we assign or revoke a role.

The assignment row in **ch4 | Source Files | protected | views | user | _form.php** should now look like the following code snippet:

```
<div class="row">
  <b>Assignments</b><br/>
  <ul class="roles">
    <?php foreach($user->assignments as $a) {
      echo $this->renderPartial('//includes/role_li',
        array(
          'user' => $user,
          'assignment' => $a,
      ));
    } ?>
  </ul>

  <?php echo $this->renderPartial('//includes/role_select',
    array(
      'user' => $user,
    ));
  ?>
</div>
```

22. Add a new JavaScript function in **ch4 | Source Files | js | user_form_ajax.js** to handle the update.

```javascript
function update_roles(updateUrl) {
  // reload the role select field
  $.ajax({
    type: 'get',
    url: updateUrl,
    data: {
      "ajax" : "1"
    },
    success: function(resp) {
      $("#role_list").replaceWith( resp );
    },
    error: function() {
      alert('Error!');
    }
  });
}
```

23. Also add a new parameter to the `revoke` function to pass in `updateUrl` and make a call to `update_roles` when a revoke is successful.

```javascript
function revoke(url, role_name, updateUrl) {
  success: function(resp) {
    $('li').remove(role);
    update_roles(updateUrl);
  },
```

24. Remove the `$(document).ready` call and move the assign function out into a named function. Pass in the values for `url` and `updateUrl`.

```javascript
function assign(url, updateUrl) {
  $.ajax({
    type: 'get',
    url: url,
    data: {
      "role" : $("#User_role").val(),
      "ajax" : "1"
    },
    success: function(resp) {
      $("ul.roles").append(resp);
      update_roles(updateUrl);
    },
    error: function() {
      alert('Error!');
    }
  });
}
```

162

25. Change the role list item view **ch4** | **Source Files** | **protected** | **views** | **includes** | **role_li.php** to include an `onClick` call to the `revoke` function and pass the `url` variables.

```php
<?php
  echo "<li id=\"role-" . $assignment->itemname. "\">" .
  $assignment->itemname .
  " <input class=\"revoke\" type=\"button\" " .
    "onClick=\"revoke('" .
      Yii::app()->controller-
        >createUrl("/user/revokeRole",
        array("id" => $user->id,
          "role_name"=>$assignment->itemname,
      "ajax"=>1)) . "', '" .
      $assignment->itemname . "', '" .
      Yii::app()->controller-
        >createUrl("/user/reloadRoles",
      array("id" => $user->id)) .
    "')\" " .
    "value=\"Revoke\" " .
  "/>" .
    "</li>";
```

26. Add the new reload action to the User controller.

```php
public function actionReloadRoles($id)
{
  if(isset($_GET['ajax'])) {
    $model=$this->loadModel($id);
    $this->renderPartial
      ('//includes/role_select',array(
      'user'=>$model,
    ), false, true);
  }
  else
  throw new CHttpException(400,'Invalid request.');
}
```

27. In the srbac interface, create an operation for `UserReloadRoles` and assign it to the `manageUser` task, as we just did for `UserAssignRole` and `UserRevokeRole`.

Now the **Assign** and **Revoke** buttons should fully work the way we expect them to.

163

Objective Complete - Mini Debriefing

In this task, we added a role assignment section to the User edit page. In the process, we created new controller actions to assign and revoke roles, and we used the srbac interface to limit the role assignment operations to the `admin` role.

Classified Intel

There is an alternate way to add operations to the system. This is particularly useful if you have added many new actions and want to create the corresponding operations all at once.

1. Go to the srbac screen.
2. Click on **Managing auth items**.
3. Click on the link named **Autocreate Auth Items** below the buttons.
4. Click on the lightbulb next to the controller that has new actions; for this task, that would have been **UserController**.
5. The list of actions without corresponding operations will be displayed. Click on the ones you want to add. You probably want to uncheck the box for **Create Tasks**, because we already have some task grouping configured.
6. You can go back to the main `AuthItem` management page to verify your new operations by clicking on the **Managing auth items** button or by clicking on the **Manage AuthItem** link underneath the buttons.

Fine-tuning Permissions

In relatively few steps, we have applied a finer grained access control to our site, but there may be one or two very tiny-grained areas that we have overlooked. In this task, we will clean up some access issues and, in the process, look at methods of applying even smaller areas of access control.

Engage Thrusters

1. One area not completely covered by access control is the comic book index. We display the list of requests. The admin, borrower, and viewer roles have access to this page, but only admin should be able to see the requests. To display the information only to authorized users, we need to add an authorization check. However, we do not want to check in the view itself, because views should not include business logic. Instead, we will perform the authorization check in the controller, within the Book View action, and pass the result to the view. Start by updating the index view (**ch4** | **Source Files** | **protected** | **views** | **book** | **index.php**) to pass a new variable named `isAdmin` to the `ListView` widget.

```
<?php $this->widget('zii.widgets.CListView', array(
  'dataProvider'=>$dataProvider,
  'viewData' => array('isAdmin' => $isAdmin),
  'itemView'=>'_view',
)); ?>
```

2. Update **ch4 | Source Files | protected | views | book | _view.php** to check this variable.

```
<?php
if ($data->requesters && $isAdmin) {
  echo "<b>Requests</b><br/>\n";
```

3. In the Book controller (**ch4 | Source Files | protected | controllers | BookController. php**), perform the access control check, set the variable, and pass it to the view.

```
$this->render('index',array(
  'dataProvider'=>$dataProvider,
  'isAdmin'=>Yii::app()->user->checkAccess('admin'),
));
```

4. We need to make the same change for the User edit screen, since we just expanded access to let users edit their own profiles. You probably don't want to allow them to choose their own level of access.

   i. Follow almost the same steps. Add the `isAdmin` field to the render call in the `User Controller Update` and `Create` actions.

   ```
   $this->render('update',array(
   'model'=>$model,
   'isAdmin'=>Yii::app()->user->checkAccess('admin'),
   ));
   ```

   ii. Pass the value to the `_form` render in the `User Update` and `User Create` views.

   ```
   <?php echo $this->renderPartial('_form', array(
     'user'=>$model,
     'person' => $model->person,
     'isAdmin' => $isAdmin,
   )); ?>
   ```

   iii. Finally, in the `User _form` view, wrap the assignments row in a check for `isAdmin`.

   ```
   <?php if ($isAdmin) { ?>
     <div class="row">
       <b>Assignments</b><br/>
   ```

```
            <ul class="roles">
              <?php foreach($user->assignments as $a) {
                echo $this->renderPartial('//includes/role_li',
                  array(
                    'user' => $user,
                    'assignment' => $a,
                ));
              } ?>
            </ul>

            <?php echo $this->renderPartial
                ('//includes/role_select',
                array(
                'user' => $user,
              ));
            ?>
          </div>
        <?php } ?>
```

5. The default authorization error message provides some details about the action that was unauthorized. What if we want to present the error with less detailed information? Start by creating a folder in the `views` directory.

    `cd ~/projects/ch4/protected/views`

    `mkdir srbac`

    We named the directory `srbac` to associate the views in the directory with the module that will use them.

6. Create a file named `access_denied.php` in the new directory.

7. For our example, we will output a simple HTML error message.

    ```
    <h2 style="color:red">
    Access denied.
    </h2>
    ```

8. Change the value of `notAuthorizedView` in the srbac configuration (**ch4** | **Source Files** | **protected** | **config** | **main.php**) to point to the view we just created.

    ```
    'notAuthorizedView'=>
      'application.views.srbac.access_denied',
    ```

9. You may have noticed that one of the tasks that we have created is named `UpdateOwnUser`. In the RBAC initialization script, the entry looks like this:

```
// user task of updating own entry
$bizRule='return (Yii::app()->user->id==Yii::app()-
  >getRequest()->getQuery('id') || Yii::app()->user->id
  == $params['id']);';
$task=$auth->createTask
  ('UpdateOwnUser','update own user entry',$bizRule);
$task->addChild('UserUpdate');
```

10. To give users access to edit their own user record, we have to create a point of entry, as usual by creating a menu item in **ch4** | **Source Files** | **protected** | **views** | **layouts** | **main.php**. We will add the entry to the top-level menu, so that our users can easily find it.

```
array(
  'label'=>'Edit Profile',
  'url'=>$this->createUrl('/user/update',
    array('id'=>Yii::app()->user->getId())),
  'authItemName' => 'UpdateOwnUser',
  'authParams' =>
  array('id'=>Yii::app()->user->getId()),
),
```

All of our roles will now have a quick link to edit the user's profile.
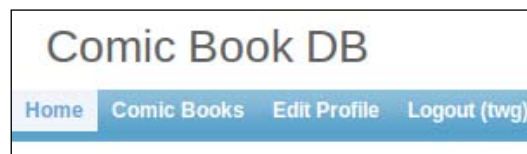
Objective Complete - Mini Debriefing

The srbac extension provides a default level of access control for each action. In this section, we demonstrated some ways to implement even finer-grained access control, such as displaying a portion of a view. We overrode a view in the module.

Classified Intel

There is an alternate way to permit users to edit their own profiles. In this task, we demonstrated the use of a business rule, but you may prefer to keep logic out of your data layer. Another way to achieve the same effect is to perform the following steps:

1. Limit access to the `UserUpdate` operation to the admin role only.

2. Create a User action named `Edit Profile` that passes the active user ID to the `Update` action.

   ```
   public function actionEditProfile()
   {
       $this->actionUpdate(Yii::app()->user->id);
   }
   ```

3. In srbac, create a new operation for `UserEditProfile` and assign it to the base role that should be able to edit its own profile.

This approach also effectively limits access to the `Update` action only to the user's own ID.

Making History

Along with access control comes audit logging. Once you grant more users access to your site, you have a greater need to record the actions users have taken. For example, say your comic book collection is so extensive that you hire some folks to help you input books. Maybe there are consistent errors in the data entry. If you have audit trails, you can identify who is making the errors, and give them more training to input the books with fewer errors. Another common use for audit trails is to retrieve an item that has been accidentally deleted.

Prepare for Lift Off

Download and unpack the `auditTrail` extension from the Yii website (`http://www.yiiframework.com/extension/audittrail/`).

Engage Thrusters

1. Copy the unpacked `auditTrail` folder into your project's modules folder.

   `cp ~/Downloads/auditTrail ~/ch4/protected/modules/.`

2. Add the module to the import array in your configuration file (**ch4** | **Source Files** | **protected** | **config** | **main.php**).

   ```
   'import'=>array(
       'application.models.*',
       'application.components.*',
   ```

```
        'application.modules.srbac.
          controllers.SBaseController',
                   'application.modules.auditTrail.models.
                    AuditTrail',
   ),
```

And to the modules array.

```
   'auditTrail'=>array(),
```

3. Replace the entry for db in **ch4 | Source Files | protected | config | console.php**.

```
   'db'=>array(
     'connectionString' =>
       'mysql:host=localhost;dbname=cbdb',
     'emulatePrepare' => true,
     'username' => 'root',
     'password' => '',
     'charset' => 'utf8',
   ),
```

4. Edit the file `yiic.php` in the project root. Change the path to `yiic` to the absolute path.

```
$yiic='/opt/lampp/htdocs/yii-1.1.10.r3566/framework/yiic.php';
```

5. Change the model and field values in **ch4 | Source Files | protected | modules | auditTrail | migrations | m110517_155003_create_tables_audit_trail.php** as follows:

```
   'model' => 'string NOT NULL',
   'field' => 'string NOT NULL',
```

Comment out the index lines for `old_value` and `new_value`.

```
   //$this->createIndex( 'idx_audit_trail_old_value',
     'tbl_audit_trail', 'old_value');
   //$this->createIndex( 'idx_audit_trail_new_value',
     'tbl_audit_trail', 'new_value');
```

6. Run the migration script supplied by the `auditTrail` extension.

```
cd ~/projects/ch4/protected
```

```
php ./yiic.php migrate --migrationPath=application.
  modules.auditTrail.migrations
```

When the script asks you if you want to apply the changes, say `Yes`.

7. In srbac, create an operation named `auditTrail@AdminAdmin` and a `manageAuditTrail` task.

8. In the **Tasks** tab under **Assign to users**, assign the `auditTrail` operation to the `manageAuditTrail` task.

9. Under the **Roles** tab, assign the `manageAuditTrail` task to the admin role.

   Access the new page at `http://localhost/cbdb/index.php/auditTrail/admin`.

10. Add the **Audit Trail Management** page to **ch4 | Source Files | protected | views | layouts | main.php** so the authority user can see the link.

    ```
    array('label'=>'AuditTrail',
      'url'=>array('/auditTrail/admin'),
    'authItemName' => 'Authority'),
    ```

11. To try out the audit capture, add the following function to the User model.

    ```php
    public function behaviors()
    {
      return array( 'LoggableBehavior'=>
      'application.modules.auditTrail.
      behaviors.LoggableBehavior', );
    }
    ```

12. Add or edit a user and then go to the Audit Trail Management page. You will see a searchable list of the changes you have made.

ID	Old Value	New Value	Action	Model	Field	Stamp	User	Model
1	tcreate	tcreater	CHANGE	User	username	2012-07-09 07:46:28	1	55

Displaying 1-1 of 1 result(s).

Objective Complete - Mini Debriefing

We installed and configured an audit trail extension to capture changes to our records.

Classified Intel

You can also add an audit trail widget to a view to show changes to an individual record. An example of this can be illustrated by adding the following snippet at the end of the User view file **ch4 | Source Files | protected | views | user | view.php**:

```php
<?php $this->widget(
  'application.modules.auditTrail.
    widgets.portlets.ShowAuditTrail',
```

```
    array(
      'model' => $model,
    )
  );
  ?>
```

Mission Accomplished

We have added two new features to the site: a library view for our users and library management utilities for us. To support these features and the different users of our system, we changed our access control scheme from the default access control filter to role-based access control. We installed the srbac extension to make management of the RBAC configuration easier, and we installed the `AuditTrail` extension to record data changes.

> Remember – if you put this site online, review your security and definitely disable Gii in the configuration.

You Ready to go Gung HO? A Hotshot Challenge

Here are some ideas to go gung ho with user functions:

- Expand your library system by adding a `Request a Book` or `Suggest a Book` function.

- Generalize the auto-complete function by creating an action extension.

- Make the Book admin page editable so that you can manage your library from one screen, instead of finding a comic book and then clicking on **edit** to update the entry.

- Create a quick entry row on the Book admin page so that you can quickly input new books with less clicking.

- Add an illustrator column to the library page.

- Create management and authorization for other entities such as publisher.

- Add a `Withdraw Request` function to the library grid.

- An alternate approach to permissions is an Access Control List implementation, which is appropriate when you want individual permissions versus group permissions. You could apply an ACL extension to the baseline chapter files to compare the different approaches.

Project 5
Service Please – Integrating Service Data

This chapter is about using third-party web service APIs in your Yii application. Usually, these interfaces are implemented as RESTful web services that use JSON for data interchange. Commonly, clients that wrap these APIs are provided in a number of programming languages. Using an API in PHP is often as simple as registering with the organization as a developer and then downloading and installing the client for PHP. As Yii provides excellent support for using third-party PHP libraries, using these services with your Yii application is easy and straightforward.

Mission Briefing

We will integrate the Google OAuth2 authentication API into Yii and then set up Google authentication. Then, we will build a comic book news stream on our site that is created from the user's Google+ stream. Then, we will set up rich data interaction via Comic Vine (`http://www.comicvine.com`).

Why Is It Awesome?

An increasingly important component of modern web development is the ability to integrate third-party web services and APIs into your application. Amazon, Facebook, Flickr, Google, Groupon, NASA, Photobucket, Reddit, Salesforce.com, Twitter, the US Postal Service, the Weather Underground, and many other organizations now provide APIs to access their data. No matter what subject matter your website spans, you will probably want to use some of the functionalities and data provided and managed by these organizations. You can use some of them for authentication, offloading the work of building and securing storage of usernames and passwords. You can integrate with popular social networking websites, so members of those sites can fully participate in interactions and ratings on your site.

Your Hotshot Objectives

- ▸ Google Me – Getting Started
- ▸ Google Me – Putting the Rubber to the Road
- ▸ Google Me – The Yii Way
- ▸ Integrating with Comic Vine – The Search, Part 1
- ▸ Integrating with Comic Vine – The Search, Part 2
- ▸ Integrating with Comic Vine – The Details
- ▸ Putting It All Together

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, follow these steps, replacing the username `james` with your own username.

1. Copy the project files into your working directory.

   ```
   cp -r ~/Downloads/project_files/Chapter\ 5/project_files
     ~/projects/ch5
   ```

2. Make the directories that Yii uses web writeable.

   ```
   cd ~/projects/ch5/
   sudo chown -R james:www-data protected/runtime assets
     protected/models protected/controllers protected/views
   ```

3. Create a link in the `webroot` directory to the copied directory.

```
cd /opt/lampp/htdocs
sudo ln -s ~/projects/ch5 cbdb
```

4. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`).

5. Create a database named `cbdb` and load the database schema (`~/projects/ch5/protected/data/schema.sql`) into it.

6. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans it is **ch5** | **Source Files** | **protected** | **config** | **main.php**).
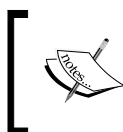
> Note that the admin login is admin/test.
>
> Also, in order to work through the first three tasks, you will need a valid Google account.

Google Me – Getting Started

We need to set up the Google OAuth 2 API and integrate it with our installation of Yii, and we need to use the Google API console to set up access to the services we want to use.

Engage Thrusters

The PHP implementation of the Google OAuth 2 API requires curl for PHP. If you are using XAMPP on Linux, it should already be enabled. You can verify this by going to `http://localhost/xampp/phpinfo.php` and looking for the section labeled `curl` or the option `--with-curl=/opt/lampp`.

1. Find `google-api-php-client` and download the latest version from the downloads page. Extract it to `protected/vendors` in your Yii directory (if the `vendors` subdirectory does not exist, create it in `protected` first).
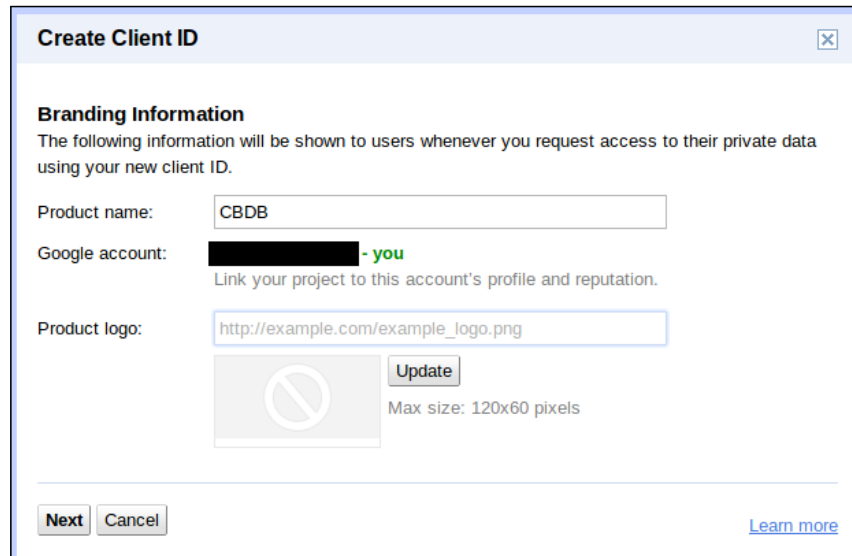
2. Go to the Google API console at `https://code.google.com/apis/console/` (you will need to log in to a valid Google account) and click on **Create project**.
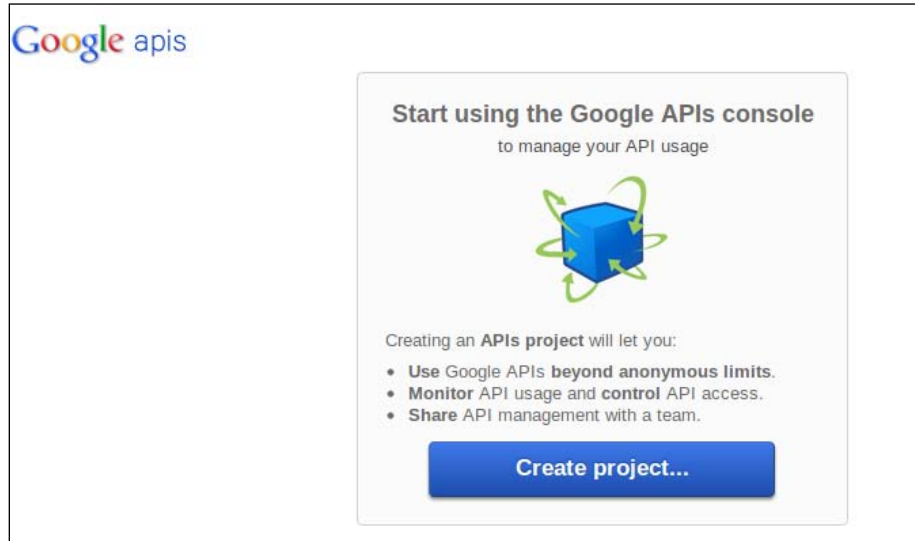
3. Put in the project name. Then, click on **API Access** and you should see something like this:
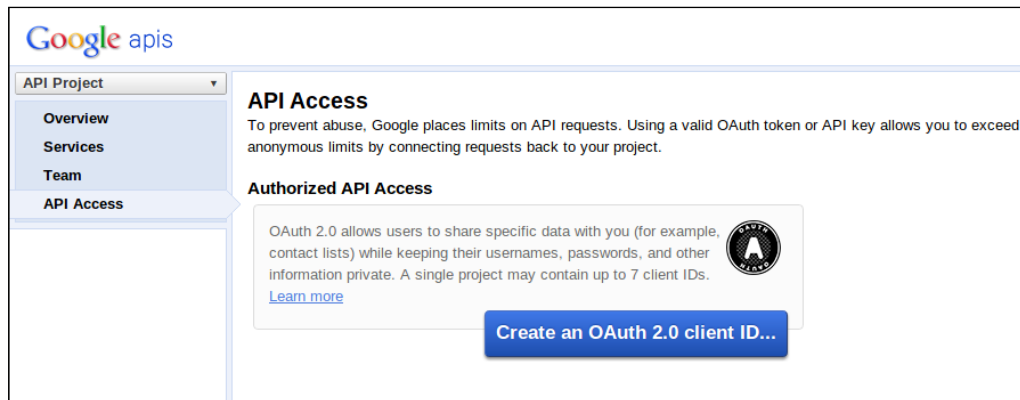
4. Click on the **Create an OAuth 2.0 client ID...** button and fill out the form.
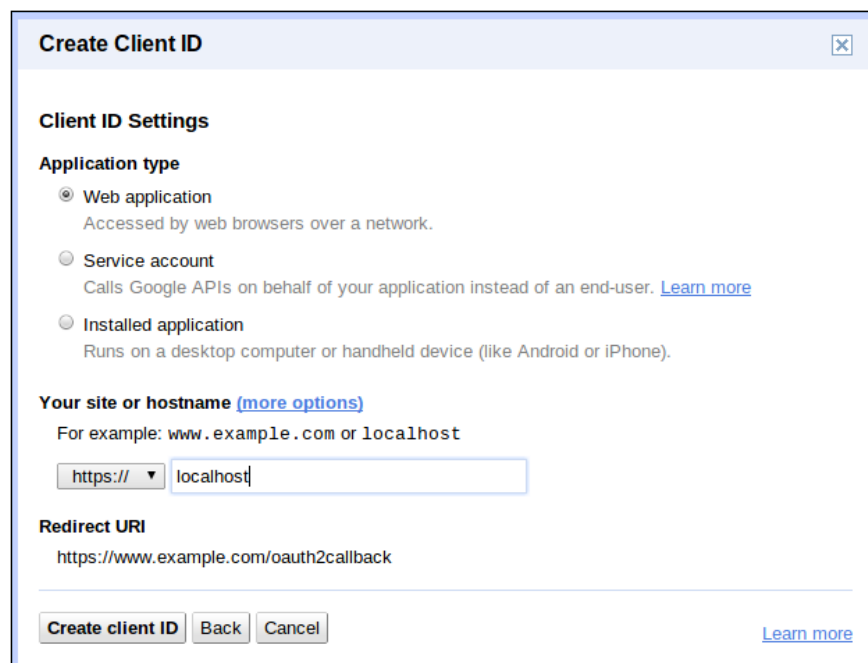
5. For now select **Web application** as the **Application type**, and then put in
   `localhost` for the hostname.

6. Once you have all that set up, go to the **Services** tab and turn on access to **Google+ API**.

7. Now we will make a couple of quick changes to our project so we can test to see if everything is set up properly.

8. Create a new controller (in `protected/controllers`) for your Google+ feed, and name it `GpfController.php`.

```php
<?php

class GpfController extends Controller
{
  public $layout='//layouts/column2';

  public function actionIndex()
  {
    $this->render('index', array());
  }
}
```

9. Modify `views/layouts/main.php` to look like this:

```php
array(
  'label'=>'Comic Books',
  'url'=>array('/book/index'),
  'items' => array(
    array('label'=>'Publishers',
      'url'=>array('/publisher/index')),
    array('label'=>'WishList',
      'url'=>array('/wish/index')),
```

```
    )
  ),
  array('label'=>'Users', 'url'=>array('/user/index')),
  array('label'=>'Google+ Feed', 'url'=>array('/gpf/index')),
  array('label'=>'Login', 'url'=>array('/site/login'),
    'visible'=>Yii::app()->user->isGuest),
  array('label'=>'Logout ('.Yii::app()->user->name.')',
    'url'=>array('/site/logout'), 'visible'=>!Yii::app()-
    >user->isGuest),
```

10. Now, add a `gpf` directory to views, and put a new `index.php` inside:

```php
<?php
  $this->breadcrumbs=array(
    'gpf',
  );
  echo "Testing, part 1.\n";
?>
```

11. Set the permissions `chown -R james:www-data views/gpf`. When you go to `http://localhost/cbdb` you should be able to click on **Google+ Feed**, and then see the testing text.

12. We are going to incorporate the Google API PHP library and see if it is installed correctly. We will call `Yii::import()` to add the vendors directory to the include path.

13. Change `index.php` to look like the following:

```php
<?php
  Yii::import('application.vendors.*');
  require_once 'google-api-php-
    client/src/Google_Client.php';
  require_once 'google-api-php-
    client/src/contrib/Google_PlusService.php';
  $this->breadcrumbs=array(
    'gpf',
  );
  echo "Testing, part 2.\n";
?>
```

14. Go to `http://localhost/cbdb` and click on **Google+ Feed**. If you don't see an error and you see the testing text, you've installed the Google Auth PHP API client correctly, and it is now working properly in your Yii environment.

Objective Complete - Mini Debriefing

We installed the Google API client for PHP. We created a Client ID with Google to allow us to access their APIs. We turned on access to the Google+ API, so that we can see our stream. We put in a mostly empty controller class, `GpfController`, with a stub for the index and we added a menu item for **Google+ Feed**. We created a small view, and put some simple smoke tests in there to see if we can use the API client in Yii.

Google Me – Putting the Rubber to the Road

Now, we will put some simple client code using the client API in the project. Everything should be in place to fetch information from our Google+ stream, so we just have to put some code in place and provide the correct values to the API for authentication.

Engage Thrusters

For this task, we'll need SSL so that we (and Google) can access our application with HTTPS. By default, it is usually enabled for XAMPP on Linux. Verify that it is working by going to `https://localhost/cbdb`. If it is not enabled, go ahead and enable it (you can find a large number of "how-to" manuals online).

1. Copy the file `index.php` in `prepared_files` over the index file in `protected/views/gpf`:

    ```
    cd ~/projects/ch5/
    cp prepared_files/gplus/index.php protected/views/gpf
    ```

2. Open the Google API console, and click on **API Access**. Then click on **Edit Settings** to the right of **Client ID for web applications**.
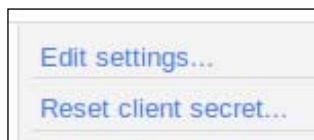
3. Then set the Redirect URI to `http://localhost/cbdb/index.php/gpf/index`:

Your final configuration will look like this:

4. Open `protected/views/gpf/index.php`.

```
$client->setClientId('CLIENT_ID');
$client->setClientSecret('CLIENT_SECRET');
$client->setRedirectUri('http://localhost/cbdb/index.php/
  gpf/index');
$client->setDeveloperKey('DEVELOPER_KEY');
```

5. Replace `CLIENT_ID`, `CLIENT_SECRET`, and `DEVELOPER_KEY` with the values from the Google API console in the **API Settings** section (`DEVELOPER_KEY` refers to the **API Key** under **Simple API Access**).

6. Now you can go to `http://localhost/cbdb` and when you click on **Google+ Feed**, it should have you connect to Google+.
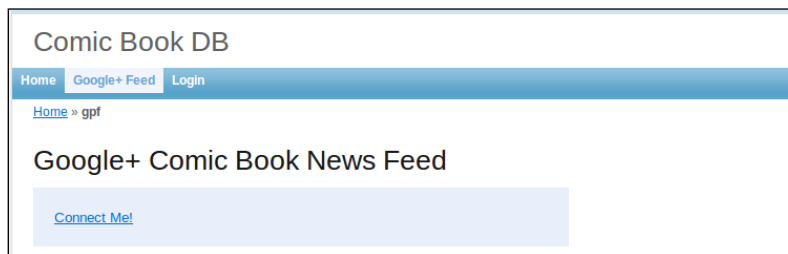
7. If you are not logged in to Google, it will ask you to authenticate:
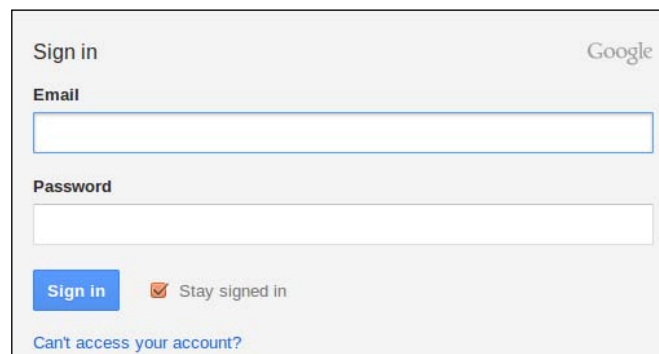
8. Go ahead and agree to have the mini-app we just wrote access your info.
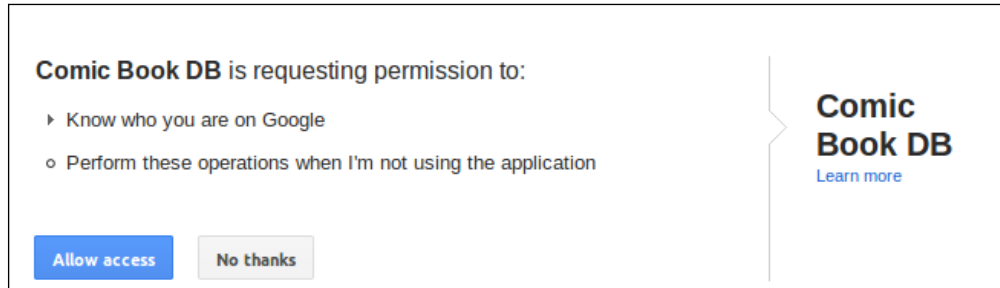
9. Then you will see a little summary of the first few entries you would see on Google+ if you searched for `#comicbooks`.

Objective Complete - Mini Debriefing

We put some simple code for fetching from our Google+ stream entirely in the view for Gpf (`views/gpf/index.php`). This code takes care of all the states for logging in to Google+ by saving the status of progression through the stages in the session. In order for this code to work, we had to set the redirect URI in the code and in the Google API console to `https://localhost/cbdb/index.php/gpf/index` (the URI of the page for viewing the Google+ feed). If these URIs do not match, you will get an error as shown in the following screenshot:
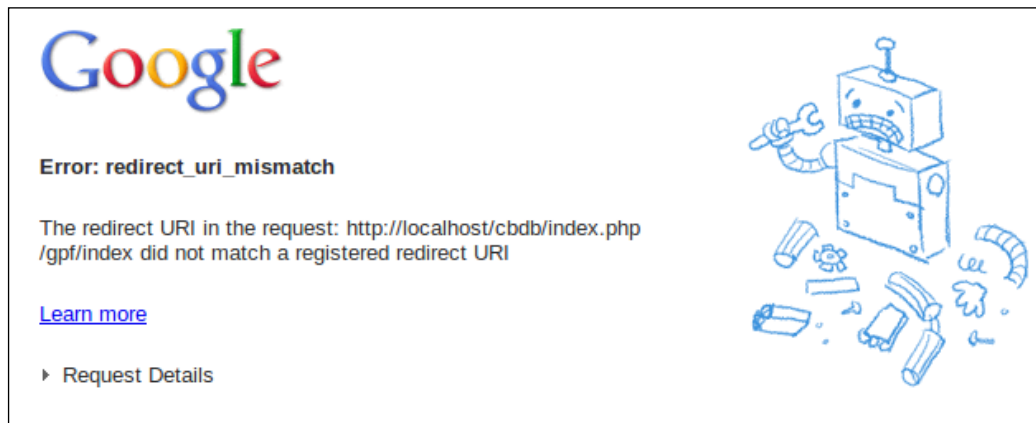
Classified Intel

The current code basically works to accomplish what we want, but we haven't implemented this using an MVC design pattern. All our logic and code is simply sitting in our view. While this was a quick and dirty way to use the API, and it hints at the power we can gain by incorporating third-party web services into our application, we haven't yet done it right. In the next task, we will do it right.

Google Me – The Yii Way

This task is about dividing the functionality implemented into the previous task into a view and a controller.

Engage Thrusters

1. Move the PHP code from the beginning of `protected/views/gpf/index.php`, going down to `<h1>Google+ Comic Book News Feed</h1>` (leave the breadcrumbs where they are) into the `indexAction()` function in `protected/controllers/GpfController.php`. Move the line initializing `$authUrl` to the top of the function. Then make some changes in the loop to store a list of entities to pass to the view, as follows (remember to replace **CLIENT_ID**, **CLIENT_SECRET**, and **DEVELOPER_KEY** with the correct values):

```php
<?php
Yii::import('application.vendors.*');
require_once 'google-api-php-client/src/apiClient.php';
require_once 'google-api-php-client/src/contrib/apiPlusService.php';

class GpfController extends Controller
{
  public $layout='//layouts/column2';

  public function actionIndex()
  {
    $authUrl = '';
    $session = Yii::app()->session;

  $client = new apiClient();
  $client->setApplicationName
    ("Google+ Comic Book News Feed");
  // Visit https://code.google.com/apis/console to
  //generate your oauth2_client_id, oauth2_client_secret,
  //and to register your oauth2_redirect_uri.
  client->setClientId('CLIENT_ID');
  $client->setClientSecret('CLIENT_SECRET');
  $client->setRedirectUri('http://localhost/cbdb/
    index.php/gpf/index');
  $client->setDeveloperKey('DEVELOPER_KEY');
  $plus = new apiPlusService($client);

  if (isset($_REQUEST['logout'])) {
    unset($session['access_token']);
  }
```

```
if (isset($_GET['code'])) {
  $client->authenticate();
  $session['access_token'] = $client->getAccessToken();
  header('Location: http://' . $_SERVER['HTTP_HOST'] .
    $_SERVER['PHP_SELF']);
}

if (isset($session['access_token'])) {
  $client->setAccessToken($session['access_token']);
}
$activityList = array();
if ($client->getAccessToken()) {
  $optParams = array('maxResults' => 100);
  $activities = $plus->activities-
    >search('#comicbooks');
  foreach($activities['items'] as $activity) {
    $activityListItem = array();
    $activityListItem['url'] = filter_var
      ($activity['url'], FILTER_VALIDATE_URL);
    $activityListItem['title'] = filter_var
      ($activity['title'], FILTER_SANITIZE_STRING,
      FILTER_FLAG_STRIP_HIGH);
    $activityListItem['content'] =
      $activity['object']['content'];
    $activityListItem['images'] = array();
      if (isset($activity['object']['attachments'])) {
        foreach($activity['object']['attachments'] as
          $attachment) {
          if ($attachment['objectType'] === 'photo') {
            $activityListItem['images'][] =
              $attachment['image']['url'];
          }
        }
      }
      $activityList[] = $activityListItem;
    }
    $session['access_token'] = $client-
      >getAccessToken();
  }
  else {
    $authUrl = $client->createAuthUrl();
  }
  $this->render('index', array('activityList' =>
    $activityList, 'authUrl' => $authUrl));
  }
}
```

2. Change the view to reflect the changes:

```php
<?php
  $this->breadcrumbs=array(
    'gpf',
  );
?>
<h1>Google+ Comic Book News Feed</h1>
<div class="box">

  <?php if(count($activityList)): ?>
  <div class="activities"><h2>Your personal comic book
    news feed: </h2>
    <?php
      foreach($activityList as $activityListItem) {
        echo("<div class='activity'><a href='" .
          $activityListItem['url'] . "'>" .
          $activityListItem['title'] . '</a><div>' .
        $activityListItem['content'] . "</div>\n");
        foreach($activityListItem['images'] as
          $imageUrl) {
          echo('<img src="' . $imageUrl . '">');
          echo("</div><div><center><img src='" .
            Yii::app()->request->baseUrl .
            "/images/hdiv.png' /></center></div>\n");
      }
    }
  ?>
  </div>
  <?php endif;
    if($authUrl) {
      print "<a class='login' href='$authUrl'>Connect
        Me!</a>";
    }
    else {
      print "<a class='logout'
        href='?logout'>Logout</a>";
    }
  ?>
</div>
```

3. Now the Google+ feed in your web app should work exactly the same way it did before we started moving the code around.

Objective Complete - Mini Debriefing

The logic responsible for communicating with the model, in this case the Google web services, now sits in the controller for our Google+ feed. The code responsible for marking up and displaying the retrieved data now sits in our view. Separating this functionality is always important when using an MVC paradigm. It makes our code more maintainable, and keeps each part of our application responsible for its own concern.

Integrating with Comic Vine – The Search, Part 1

Comic Vine's mission statement is "to be the most useful and easy to use comic book website in the world". They provide a nice API with JSON or XML formats for integration. Without special arrangement, their API is only for personal use. If you want to take your website to the masses while using their stuff, be sure to give them a call first. Comic Vine does not provide a way to look up comic books by ISSN, so we are going to implement a search by title.

Engage Thrusters

The chapter files include a simple PHP wrapper for the Comic Vine functionality that we will use.

We are going to briefly discuss how this wrapper works. Here is the wrapper:

```php
<?php

class CbdbComicVine {
  private $apiKey;
  private $baseUrl;

  public function __construct($apiKey, $baseUrl =
    'http://api.comicvine.com/') {
    $this->setApiKey($apiKey);
    $this->setBaseUrl($baseUrl);
  }

  public function setAPiKey($apiKey) {
    $this->apiKey = $apiKey;
  }
```

```php
public function setBaseUrl($baseUrl) {
  $this->baseUrl = $baseUrl;
}

public static function buildQueryString($paramArray) {
  $paramString = http_build_query($paramArray);
  ;

  if ($paramString) {
    $paramString = '?' . $paramString;
  }
  return $paramString;
}

public static function makeRequest($url, $paramArray) {
  $curl = curl_init();
  curl_setopt($curl, CURLOPT_URL, $url .
    CbdbComicVine::buildQueryString($paramArray));
  curl_setopt($curl, CURLOPT_HEADER, false);
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  curl_setopt($curl, CURLOPT_HTTP_VERSION,
    CURL_HTTP_VERSION_1_1);
  $response = curl_exec($curl);
  $status = curl_getinfo($curl);
  if ($status['http_code'] == 200) {
    $responseObject = json_decode($response);
    if (is_object($responseObject) &&
      ($responseObject->status_code == 1)) {
      return array('error' => 0, 'content' => $responseObject);
    }
    return array('error' => 1, 'error_type' => 'site', 'content'
      => $responseObject);
  }
  return array('error' => 1, 'error_type' => 'transfer',
    'content' => $status);
}

private function setInitParams(&$params) {
  $params['api_key'] = $this->apiKey;
  $params['format'] = 'json';
  return $params;
}
```

```php
public function getNewParamsArray() {
  $params = array();
  return $this->setInitParams($params);
}

public function baseRequest($resource, $addlParams) {
  $params = $this->getNewParamsArray();
  $params = array_merge($params, $addlParams);
  return $this->makeRequest($this->baseUrl . $resource . '/',
    $params);
}

public function detailRequest($resource, $id, $addlParams) {
  $params = $this->getNewParamsArray();
  $params = array_merge($params, $addlParams);
  return $this->makeRequest($this->baseUrl . $resource . '/' .
    $id . '/', $params);
}

public function volumeSearch($query, $params = array(), $offset
  = 0, $limit = 20) {
  $params['query'] = $query;
  $params['resources'] = 'volume';
  $params['offset'] = $offset;
  $params['limit'] = $limit;

  return $this->baseRequest('search', $params);
}

public function issuesForVolume($volumeId, $params = array()) {
  $params['field_list'] = 'issues';
  return $this->detailRequest('volume', $volumeId, $params);
}


public function volume($id, $params = array()) {
  return $this->baseRequest("volume/$id", $params);
}

public function issue($id, $params = array()) {
  return $this->baseRequest("issue/$id", $params);
}
}
```

The Comic Vine API works like most web services. You create an API key and then use the key to make requests. In the previous object, it is stored in the private data member `$apiKey`. It is set via the constructor. By default, `$baseUrl` is set to `http://api.comicvine.com`, also via the constructor. The member function `buildQueryString()` takes a parameter array, populated with name-value pairs, and produces a URL query string. The member function `makeRequest()` accepts a URL and a parameter array, encodes the parameters as a URL query string, makes the queries using `curl`, and then decodes the JSON reply and returns it in native PHP data constructs. It wraps the results in an associative array containing error information in a standard format. I use `getNewParamsArray()` to initialize an array to set the API key and to set the format to JSON.

The functions `buildQueryString()`, `makeRequest()`, and `getNewParamsArray()` are not intended to be used directly. They are utility functions used in combination in `baseRequest()` and `detailRequest()`, which can be used to implement most, if not all, of the functionality offered by Comic Vine. The function `baseRequest()` accesses resources the function and `detailRequest()` accesses resource details. In order to search for particular volumes and issues, we have implemented `volumeSearch()`, `volume()`, `issue()`, and `issuesForVolume()`. At this point, it is obviously trivial to implement functionality found in the API documentation for Comic Vine. Now that we have discussed the wrapper, we are going to integrate it with our website.

Let's create a controller for our search. Create `protected/controller/CvController. php` and insert the following code snippet:

```php
<?php
Yii::import('application.vendors.*');
require_once 'comicvine/comicvine.php';

class CvController extends Controller
{
  public $layout='//layouts/column2';

  public static function newCv()
  {
    return new CbdbComicVine
      ('39aed1911b2cbffd08f19b4bf5922fd96ccf3b4f');
    //Replace this with your API key
  }

  public function actionIndex()
  {
    $this->redirect(array('search'));
  }

  public function actionSearch()
  {
```

```
      $itemList = array();
      $cv = $this->newCv();
      if (isset($_GET['search'])) {
        $offset = 0;
        if (isset($_GET['offset'])) {
          $offset = $_GET['offset'];
        }
        $result = $cv->volumeSearch($_GET['q'], array(), $offset);
        $this->render('search', array('model'=>$model,
          'result' => $result, 'q' => $_GET['q']));
        }
      else {
          $this->render('search',array('model'=>$model));
          }
    }
  }
```

Typically, we would move things like the API key into a config file. For simplicity we will leave it here.

Let's look at what's going on in this controller:

1. The function `actionSearch()` takes whatever is passed in the `'q'` parameter and conducts a volume search on it. If offset is set in the query string, it is passed along to `volumeSearch()` as a parameter, otherwise it defaults to `0`. Let's make a simple view to test this in `protected/views/cv/search.php`:

    ```php
    <?php

    $this->breadcrumbs=array(
      'Comicvine',
      'Search'
    );

    if (isset($result)) {
      if (isset($result['content'])) {
        if (isset($result['content']->results)) {
          foreach($result['content']->results as $rec){
            echo("<div class = 'result_row'>\n");
            echo("<div class = 'result_name'>\n");
            echo("$rec->name\n");
            echo("</div>\n");
            echo("<div class = 'result_details'>\n");
            echo("<a href='" . $rec->site_detail_url . "'
              target='_blank'>". 'View Details' . "</a>\n");
            echo("</div>\n");
            echo("</div>\n");
          }
    ```

```
        }
      }
    }

    ?>
```

2. Let's see where that leaves us. Search for Spiderman by entering the URL `http://localhost/cbdb/index.php/cv/search?search&q=Spiderman`.
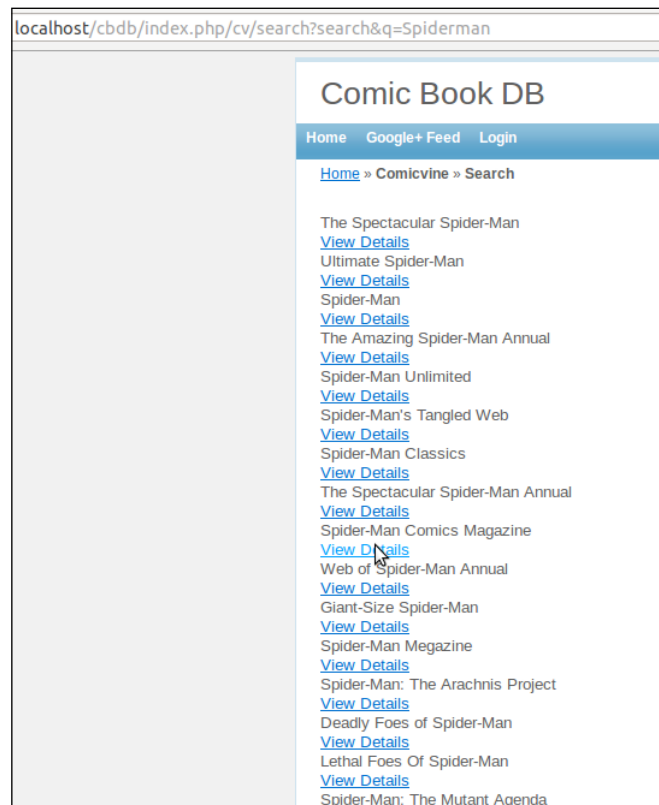
You can click on **View Details** and it will open a new tab with the Comic Vine details for each result. Our search has a couple of problems. It only shows the first 20 results, and it's really not good. Let's fix the second problem so it looks a little nicer while we work on pagination. Add some styling at the very top of the file and then add a couple of lines.

```css
<style type="text/css">
  .result_header {
    text-align: center;
    font-size: 150%;
```

193

```css
      text-decoration: underline;
      width: 350px;
      padding: 4px;
    }

    .result_name {
      font-weight:bold;
      float: left;
      width: 250px;
    }

    .result_details {
      float: left;
      width: 100px;
    }

    .result_row {
      float: left;
      border-width: 1px;
      border-color: #00a;
      border-style: solid;
      padding: 4px;
    }

    .clear_left {
      clear: left;
    }
</style>
```
```php
<?php

$this->breadcrumbs=array(
  'Comicvine',
  'Search'
);




if (isset($result)) {
  if (isset($result['content'])) {
    if (isset($result['content']->results)) {
        echo "<div class = 'result_header'>Results</div>";
      foreach($result['content']->results as $rec){
        echo("<div class = 'result_row'>\n");
        echo("<div class = 'result_name'>\n");
```

194

```
        echo("$rec->name\n");
        echo("</div>\n");
        echo("<div class = 'result_details'>\n");
        echo("<a href='" . $rec->site_detail_url . "'
          target='_blank'>". 'View Details' . "</a>\n");
        echo("</div>\n");
        echo("</div>\n");
        echo("<div class='clear_left' />");
      }
    }
  }
}
?>
```
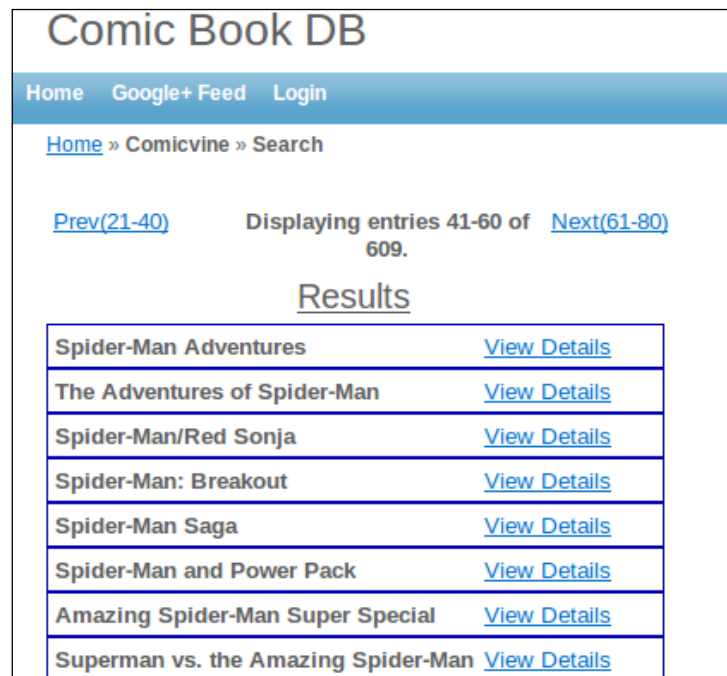
Much better. Typically, we would put the embedded styles in a separate CSS file, but we will put it all in this file for simplicity.

Results	
The Spectacular Spider-Man	View Details
Ultimate Spider-Man	View Details
Spider-Man	View Details
The Amazing Spider-Man Annual	View Details
Spider-Man Unlimited	View Details
Spider-Man's Tangled Web	View Details
Spider-Man Classics	View Details
The Spectacular Spider-Man Annual	View Details
Spider-Man Comics Magazine	View Details
Web of Spider-Man Annual	View Details
Giant-Size Spider-Man	View Details
Spider-Man Megazine	View Details
Spider-Man: The Arachnis Project	View Details
Deadly Foes of Spider-Man	View Details
Lethal Foes Of Spider-Man	View Details
Spider-Man: The Mutant Agenda	View Details
Spider-Man: Friends and Enemies	View Details
Spider-Man: Web Of Doom	View Details
Spider-Man and X-Factor: Shadowgames	View Details
Spider-Man: Hobgoblin Lives	View Details

Now let's fix the pagination. This will take a little more work. We will write a function to create a pagination bar, complete with links for the next and previous results. Put this at the top of the view, right below the opening PHP tag.

```php
function printPagination($result, $q) {
  if (isset($result['content']->offset) &&
    isset($result['content']->limit)) {
    $totalResults = isset($result['content']-
      >number_of_total_results)?$result['content']-
      >number_of_total_results:0;
    echo("<div class = 'pagination_row'>\n");
    echo("<div class = 'left_pagination'>\n");
    if ($result['content']->offset != 0) {
      $lowerLimit = ($result['content']->offset -
        $result['content']->limit >= 0) ?
          $result['content']->offset -
            $result['content']->limit : 0;
      $upperLimit = $result['content']->offset - 1;
      $upperLimit  = ($upperLimit > $totalResults -
        1)?$totalResults - 1:$upperLimit;
      echo("<a href='" . Yii::app()->request-
        >getBaseUrl() . '/' . Yii::app()->request-
          >getPathInfo() .
        "?search=1&offset=$lowerLimit&q=$q'>Prev
          (" . ($lowerLimit + 1) .  '-' .
            ($upperLimit + 1) . ")</a>\n");
    }
    echo("</div>");


    echo("<div class = 'center_pagination'>\n");
    $lowerLimit = $result['content']->offset;
    $upperLimit = $result['content']->offset +
      $result['content']->limit - 1;
    $upperLimit  = ($upperLimit > $totalResults -
      1)?$totalResults - 1:$upperLimit;
    echo('<center>');
    if ($totalResults == 0) {
      echo('Displaying entries 0-0 of 0.');
    }
    else {
      echo('Displaying entries ' . ($lowerLimit + 1) . '-
        ' . ($upperLimit + 1) . " of $totalResults.");
    }
    echo('</center>');
```

```
            echo("</div>");

            $lowerLimit = $result['content']->offset +
               $result['content']->limit;
            if ($lowerLimit < $totalResults - 1) {
               $upperLimit = $result['content']->offset + 2 *
                  $result['content']->limit - 1;
               $upperLimit = ($upperLimit > $totalResults -
                  1)?$totalResults - 1:$upperLimit;
               echo("<div class = 'right_pagination'>\n");
               echo("<a href='" . Yii::app()->request-
                  >getBaseUrl() . '/' . Yii::app()->request-
                     >getPathInfo() .
                  "?search=1&offset=$lowerLimit&q=$q'>Next(" .
                     ($lowerLimit + 1) . '-' . ($upperLimit + 1) .
                     ")</a>\n");
               echo("</div>");
            }
            echo("</div>\n");
         }
      }
```

3. This function simply takes the information in the result and prints a link to the next and previous results if any. Add these two lines and try it out:

```
if (isset($result)) {
   if (isset($result['content'])) {
      printPagination($result, $q);
      if (isset($result['content']->results)) {
         echo "<div class = 'result_header'>Results</div>";
         foreach($result['content']->results as $rec){
            echo("<div class = 'result_row'>\n");
            echo("<div class = 'result_name'>\n");
            echo("$rec->name\n");
            echo("</div>\n");
            echo("<div class = 'result_details'>\n");
            echo("<a href='" . $rec->site_detail_url . "'
               target='_blank'>". 'View Details' . "</a>\n");
            echo("</div>\n");
            echo("</div>\n");
            echo("<div class='clear_left' />");
         }
      }
      printPagination($result, $q);
   }
}
```

197

4. Now add the following to the style section to make it look better:

```
.left_pagination {float: left; width: 100px;}
.center_pagination {font-weight:bold; float: left; width: 190px;}
.right_pagination {float: left; width: 100px;}
.pagination_row {float: left; padding: 4px;}
```

5. Be sure to add a div to clear the left float after the pagination row (add this line near the end of `printPagination()`).

```
echo("</div>\n");
echo("<div class='clear_left' />");
```

6. You will wind up with the following result:

The view displays 20 results per page, with the pagination row at the top and bottom of each page.

Objective Complete - Mini Debriefing

We covered a lot of material in this task. If the web service you are trying to use does not provide an API wrapper in a language you can use, you will have to write your own. This task has demonstrated some basic tactics to accomplish this. Then, we integrated a wrapper with a Yii controller. Pagination is often a necessary task, and we see how to deal with it in a relatively standard way when using Comic Vine. Yii has a component for handling pagination information called `CPagination`, but since we have already handled this on our own as an educational exercise, we won't use it for this task.

In the next task, we will add a search form at the top of the screen, so you don't have to use the URL parameter to search.

Integrating with Comic Vine – The Search, Part 2

We have laid the groundwork for our Comic Vine volume search. Now we will add a search form. We will use the `CActiveForm` widget.

Engage Thrusters

1. Create a model for the widget to use. Create the file `protected/models/CvSearchForm.php` with the following contents:

```php
<?php
class CvSearchForm extends CFormModel {
  public $query;

  public function rules()
  {
    return array(
      array('query', 'required'),
    );
  }

  public function attributeLabels()
  {
    return array(
      'query'=>'Title Search',
    );
  }
}
```

Our needs from the model are minimal and this should take care of it.

2. Now add this to the controller (be sure to change the `if` condition to an `elseif`).

```
$model = new CvSearchForm();
if (isset($_POST['CvSearchForm']))
{
  $itemList = array();
       $model->attributes = $_POST['CvSearchForm'];
  if ($model->validate()) {
    $result = $cv->volumeSearch($model->query);
    $this->render('search',array(
      'model' => $model,
      'itemList' => $itemList,
      'result' => $result,
      'q' => $model->query
    ));
  }
}
elseif (isset($_GET['search'])) {
  $offset = 0;
  if (isset($_GET['offset'])) {
    $offset = $_GET['offset'];
```

`CActiveForm` is used and explained extensively in *Project 3, Access All Areas – Users and Logins*. So if you don't know what's going on at this point you might want to refer to it.

3. In the view, add the following lines of code immediately after the breadcrumbs:

```
$this->breadcrumbs=array(
  'Comicvine',
  'Search'
);

$form=$this->beginWidget('CActiveForm', array(
  'id'=>'search-form',
  'enableClientValidation'=>true,
  'clientOptions'=>array(
    'validateOnSubmit'=>true,
  ),
));
echo $form->errorSummary($model);
?>
<div class="row">
  <?php echo $form->labelEx($model,'query'); ?>
  <?php
  if (isset($q)) {
```

```
    echo $form->textField($model,'query',array
       ('size'=>40, 'value' => $q));
  }
  else {
     echo $form->textField($model,'query',array('size'=>40));
  }
  ?>
  <?php echo $form->error($model,'query'); ?>
</div>
<div class="row buttons">
  <?php echo CHtml::submitButton('Submit', array('size'=>40)); ?>
</div>
<?php $this->endWidget();
if (isset($result)) {
```

Now we have a pretty good interface for our volume search.

Objective Complete - Mini Debriefing

In just a few lines of code, we were able to add our search box. We put the value of q in the textbox if the q parameter is set, because this is how the link for the next and previous links are constructed, and we still want to show the query for the search in the blank.

Integrating with Comic Vine – The Details

We have a way to search for volumes, which are collections of issues. However, we don't have a way to browse and select individual issues. Let's fix that.

Engage Thrusters

We need to start by making an action for listing issues in CvController. Then, we will make a view. Sounds familiar? Let's go.

1. Change `CvController` to look like the following code snippet:

```php
<?php
Yii::import('application.vendors.*');
require_once 'comicvine/comicvine.php';

class CvController extends Controller
{
  public $layout='//layouts/column2';

  public static function newCv()
  {
    return new  CbdbComicVine
      ('39aed1911b2cbffd08f19b4bf5922fd96ccf3b4f');
    //Replace this with your API key
  }

  public static function errorHandler($result, $view) {
    if ($result['error']) {
      $this->render($view, array('error'=>
       $result['content']->error));
      return true;
    }
    return false;
  }
```

```php
public function actionIndex()
{
  $this->redirect(array('search'));
}

public function actionSearch()
{
  $itemList = array();
  $cv = $this->newCv();
  $model = new CvSearchForm();
  if (isset($_POST['CvSearchForm']))
  {
    $itemList = array();
        $model->attributes = $_POST['CvSearchForm'];
    if ($model->validate()) {
      $result = $cv->volumeSearch($model->query);
      $this->render('search',array(
        'model' => $model,
        'itemList' => $itemList,
        'result' => $result,
        'q' => $model->query
      ));
    }
  }
  elseif (isset($_GET['search'])) {
    $offset = 0;
    if (isset($_GET['offset'])) {
      $offset = $_GET['offset'];
    }
    $result = $cv->volumeSearch($_GET['q'], array(),
      $offset);
    $this->render('search', array('model'=>$model,
      'result' => $result, 'q' => $_GET['q']));
      }
  else {
      $this->render('search',array('model'=>$model));
       }
}

public function actionIssues() {
  $cv = $this->newCv();
  $title = '';
  if (isset($_GET['title'])) {
    $title = CHtml::encode($_GET['title']);
```

```
        }
        if (isset($_GET['volume_id'])) {
          $volumeId = $_GET['volume_id'];
          $result = $cv->issuesForVolume($volumeId);
          if (!$this->errorHandler($result, 'issues')) {
            $issues = $result['content']->results->issues;
              $this->render('issues',array('result' => $issues,
                'title' => $title));
          }
        }
        else {
            $this->render('issues',array('error'=>'No volume id or
              issue id specified', 'result' => null));
        }
      }
    }
}
```

This gives us a way to fetch the issues associated with a volume ID specified in the query string as `volume_id`. If title is specified, we pass it through, escaping HTML special characters. We need to see what we are doing, so make a view in the file `protected/views/cv/issues.php`:

```php
<?php
$this->breadcrumbs=array(
  'Comicvine',
);
if (isset($error)) {
  echo($error);
}
else {
?>
<style type="text/css">
  .search_row {border-width: 1px; border-color: #0000aa;
    border-style: solid; padding: 4px;}
  .issue_number {float: left; width: 50px;}
  .issue_name {float: left; width: 250px; font-weight:
    bold;}
  .issue_detail {float: left; width: 50px;}
</style>
<div class='search_header'>
<?php
```

```php
      if(isset($title)) {
        echo("<center><u><h3>$title</h3><u></center>");
      }
    ?>

    </div>
    <?php
      foreach($result as $issue) {
        echo("<div class='search_row'>\n");
        echo("<div class='issue_number'>\n");
        echo((int) $issue->issue_number);
        echo("</div>");
        echo("<div class='issue_name'>\n");
        echo($issue->name?CHtml::encode
          ($issue->name):' ');
        echo("</div>");
        echo("<div class='issue_detail'>\n");
        echo("<a href='" . Yii::app()->request->getBaseUrl()
          . '/' . Yii::app()->request->getPathInfo() .
             '?issue_id='  . $issue->id . "'
               target='_blank'>Details </a>");
        echo("</div>");
        echo("<div style='clear: left;'></div>");
        echo("</div>\n");
      }
    }
    ?>
```

2. If you happen to have a valid volume ID for Comic Vine, you can now test with the URL `http://localhost/cbdb/index.php/cv/issues?volume_id=2870`. It does not sort by issue number, so let's fix that.

3. Add a sorting function to the controller:

```php
    static function sortIssues($a, $b)
     {
     $l = $a->issue_number;
     $r = $b->issue_number;
        if ($l == $r) {
              return 0;
          }
         return ($l > $r) ? +1 : -1;
     }
```

4. Now sort the array in the action:

```
$result = $cv->issuesForVolume($volumeId);
if (!$this->errorHandler($result, 'issues')) {
  $issues = $result['content']->results->issues;
  usort($issues, array('CvController', 'sortIssues'));
    $this->render('issues',array('result' =>
      $issues, 'title' => $title));
}
```

The list is now sorted but the details link does not work.

```
if (isset($_GET['volume_id'])) {
  $volumeId = $_GET['volume_id'];
  $result = $cv->issuesForVolume($volumeId);
  if (!$this->errorHandler($result, 'issues')) {
    $issues = $result['content']->results->issues;
    usort($issues, array('CvController',
     'sortIssues'));
      $this->render('issues',array
        ('result' => $issues, 'title' => $title));
  }
}
elseif (isset($_GET['issue_id'])) {
    $issueId = $_GET['issue_id'];
    $result = $cv->detailRequest('issue', $issueId, array());
    if (!$this->errorHandler($result, 'issues')) {
      $this->redirect($result['content']->results-
        >site_detail_url);
    }
}
else {
```

5. Add the previous code snippet and the site will redirect to the Comic Vine site detail URL, after fetching the detail information for that particular issue.

Objective Complete - Mini Debriefing

Now we have a way to list issues for a particular volume. We will tie this to the volume search in the next task.

Putting It All Together

We will make a link on each volume in the volume search that shows the issues in that volume.

Engage Thrusters

At this point, all we have to do to accomplish this is to make some changes in the view.

1. Change `protected/views/cv/issues.php`:

```
foreach($result['content']->results as $rec){
    echo("<div class = 'result_row'>\n");
    echo("<div class = 'result_name'>\n");
    echo("$rec->name\n");
    echo("</div>\n");
    echo("<div class = 'result_details'>\n");
    echo("<a href='" . $rec->site_detail_url .
     "' target='_blank'>". 'View Details' . "</a>\n");
    echo("</div>\n");
    echo("<div class = 'result_issues'>\n");
    echo("<a href='" . Yii::app()->request->getScriptUrl() .
       '/cv/issues?title=' . urlencode($rec->name) .
       '&volume_id=' . $rec->id .
      "' target='_blank'>Issues(" .
       $rec->count_of_issues . ')</a>');
    echo("</div>\n");
    echo("</div>\n");
    echo("<div class='clear_left' />");
}
```

2. Fix the styling at the top of the file for the new field.

```
.result_issues {float: left; width: 100px;}
```

3. Change the width of `.result_header` from `350px` to `450px`.

Objective Complete - Mini Debriefing

Now the volume search has a column for issues that opens a new tab with a list of issues and the volume title at the top. We have now successfully wrapped a hierarchical collection of resources in Comic Vine and incorporated it into Yii.

Mission Accomplished

We implemented a volume search and an issue browser, both integrated with Comic Vine.

Here is what the search looks like:

You Ready to go Gung HO? A Hotshot Challenge

This project has demonstrated how to incorporate a third party API into your Yii web app. Take this information to the next level by implementing either a way to tie a Comic Vine issue to an issue in our database or a way to import an issue from Comic Vine into our database to complete the integration.

Project 6

It's All a Game

All work and no play makes James a dull boy. Game programming evokes a slightly different mindset than developing business applications. We will use these exercises to gain a new perspective. We can learn Yii and have fun too! In this chapter, we will try to prove that.

Mission Briefing

We will make two games that the user can play for fun and practice. We will leverage Yii to do this quickly with a small amount of code.

Why Is It Awesome?

Implementing a game with a development framework like Yii can be challenging and rewarding. If you implement it using the MVC model, you have to come up with a stateful model that makes sense. If you don't take the correct precautions, it is easy for the users to cheat. We can learn a lot by exploring the concepts involved.

Your Hotshot Objectives

- ▶ Updating the Database and Running Gii for Hangman
- ▶ Creating a JSON Endpoint for Hangman
- ▶ Developing the Controller – Creating the DB Entry
- ▶ Developing the Controller – Making the Rules
- ▶ Developing the View
- ▶ Improving the View
- ▶ Authorized Entry Only
- ▶ Reusing Code – Making a New Game

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, follow these steps, replacing the username `james` with your own username.

1. Copy the project files into your working directory.

   ```
   cp –r ~/Downloads/project_files/Chapter\ 6/project_files
     ~/projects/ch6
   ```

   > The source files can be downloaded from the *Support* page at `http://www.packtpub.com/support`.

2. Make the directories that Yii uses web writeable.

   ```
   cd ~/projects/ch6/
   ```

   ```
   sudo chown -R james:www-data protected/runtime assets
     protected/models protected/controllers protected/views
   ```

3. Create a link in the `webroot` directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   ```

   ```
   sudo ln -s ~/projects/ch6 cbdb
   ```

4. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`).

5. Create a database named `cbdb` and load the database schema (`~/projects/ch6/protected/data/schema.sql`) into it.

6. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans: **ch6 | Source Files | protected | config | main.php**).

7. This project requires curl for PHP. If you are using XAMPP on Linux, it should already be enabled. You can verify this by going to `http://localhost/xampp/phpinfo.php` and looking for the section labeled `curl` or the option `--with-curl=/opt/lampp`. If it is not enabled, you will need to determine how to enable it for your particular environment.

   > Note that the admin password is `test`.

Updating the Database and Running Gii for Hangman

We're going to develop Hangman. The game will work as follows. A suitable title will be randomly selected from the database. It will be drawn as blanks and the user can begin guessing letters. If the letter is present, it will be filled in. Each time the user misses a letter, a new piece of the man is hung. A list of guessed letters will be kept. No letter can be guessed twice. If the user misses six letters, the entire man is hung and the game is over. We need a place to persist information about games. So in this task, we will create a database table and run Gii to create a model and a controller for the entity.

Prepare for Lift Off

Look in `protected/config/main.php` and make sure that the srbac `debug` parameter is set to true.

```
'srbac' => array(
  'userclass'=>'User', //default: User
  'userid'=>'id', //default: userid
  'username'=>'username', //default:username
  'delimeter'=>'@', //default:-
  'debug'=>true, //default :false
  'pageSize'=>10, // default : 15
  'superUser' =>'Authority', //default: Authorizer
  'css'=>'srbac.css', //default: srbac.css
```

Engage Thrusters

1. Connect to the `cbdb` database and run the following command to create the `hangman` table:

```
CREATE TABLE `hangman` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `title` varchar(80) NOT NULL,
  `guessed` varchar(26) NOT NULL default '',
  `fails` tinyint(3) unsigned default 0,
  `token` varchar(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE(`token`)
) ENGINE=InnoDB;
```

2. Now run the model generator with Gii and point it to the `hangman` table, and generate a model named `Hangman`:

Table Prefix
[empty]
Table Name *
hangman

Model Class *
Hangman

Base Class *
CActiveRecord

3. Click on **preview** and then on **generate** and you should see the following output:

```
Generating code using template "/opt/lampp/htdocs/
   yii-1.1.10.r3566/framework/gii/generators/model/
   templates/default"...
 generated models/Hangman.php
done!
```

4. Now generate a controller with the controller generator in Gii. Input `hangman` for the controller ID, and `create play` for the action IDs:

Controller Generator

This generator helps you to quickly generate a new controller class, one or several controller actions and their corresponding views.

*Fields with * are required. Click on the* highlighted fields *to edit them.*

Controller ID *
hangman

Base Class *
Controller

Action IDs
create play

Code Template *
default (/opt/lampp/htdocs/yii-1.1.10.r3566/framework/gii/generators/controller/templates/default)

5. Click on **preview** and then on **generate** and you should see the following output:

```
Generating code using template "/opt/lampp/htdocs/yii-
  1.1.10.r3566/framework/gii/generators/controller/
  templates/default"...
 generated controllers/HangmanController.php
 generated views/hangman/create.php
 generated views/hangman/play.php
done!
```

Objective Complete - Mini Debriefing

We have created a database table called `Hangman` with the following columns: `title`, `guessed`, `fails`, and `token`. Each row in the table will represent a single round of hangman. The table `title` will store the title of the book the user is trying to guess, `guessed` will be a string containing all the letters that have been guessed (in alphabetical order), and `fails` will store the number of guesses the user has got wrong. A unique lookup string (a token) that is unrelated to the ID of the record will be stored in the table `token`. The token should be constructed in such a way that it is very difficult (virtually impossible) to guess. We generated a model to encapsulate the `hangman` table as a `Hangman` object. We have generated a controller, named `HangmanController.php`, with two actions: `play` and `create`. The action `create` will be used to select a title and create a token, at which point it will redirect to `play`. `play` should use the token to statefully track the round of hangman. A view for each action has been added. The plan is to use the `create` view to display errors related to creating a new game, and to develop the view for our game in the `play` view. Hold these thoughts while we take a very short detour in the next task.

Creating a JSON Endpoint for Hangman

We need a way to fetch all the titles of our books, so we can randomly select one for hangman. We could fetch them all using the book model, but have chosen to instead expose this as a JSON endpoint. In this case, we will be fetching the list of books from the controller using the `curl` library, but we could just as easily use AJAX to fetch them from the view, or this endpoint could be the start of a web service API we could expose to third parties that would like to use our data outside our application.

Engage Thrusters

1. Open `protected/controllers/BookController.php` and add the following lines after the other actions:

```
public function actionTitlelist()
   {
      header('Content-type: application/json');
      $books = Book::model()->findAll();
      $ret = array();
      foreach ($books as $book) {
        $ret[] = $book['title'];
      }
      echo CJSON::encode($ret);
      Yii::app()->end();
   }
```

2. Open a browser and navigate to `http://localhost/cbdb/index.php/book/titlelist`. You should see something like the following screenshot:
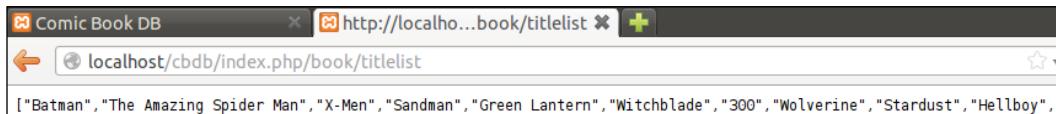
Objective Complete - Mini Debriefing

This very short task has shown an ad-hoc way to produce useful JSON endpoints. Because the action in the controller calls echo, it prints it in the view, even though there is no render (and no view file).

Classified Intel

There are other ways to accomplish this. You could just put the following code in the `Hangman` controller we are about to write, in the `create` action where we will fetch the list of titles:

```
$books = Book::model()->findAll();
$titles = array();
foreach ($books as $book) {
  $titles[] = $book['title'];
}
```

Then, you would not have to use `curl` to fetch the list.

It could be argued that serving JSON data directly through a controller does not strictly adhere to the MVC philosophy, because it is missing a view. If you still want to generate JSON, but you want to explicitly define a view, you can do that as well, by making an appropriate barebones layout and using that in the view. In that case you would call `render()` in the usual way, and then convert the passed data to JSON in the view.

Developing the Controller – Creating the DB Entry

We will put code in the `create` action for the `Hangman` controller to do four basic things:

1. Pick a title at random.

2. Generate a token.

3. On success, create the record, and redirect to play, passing the token as a parameter.

4. On failure, display the error in the view.

Engage Thrusters

1. Open `protected/controllers/HangmanController.php` and add the following function at the top of the `HangmanController` class:

   ```
   private function errorAndEnd($action, $error) {
     $this->render($action, array('error' => $error));
     Yii::app()->end();
   }
   ```

   When called, this will pass `error` to the view and cause the application to terminate.

2. Then add code to `actionCreate()` so it looks like the following:

   ```
   public function actionCreate() {
     $error = '';
     $request = Yii::app()->request;
     $jsonUrl = $request->hostInfo . $request->baseUrl .
       '/index.php/book/titlelist';
     $ch = curl_init($jsonUrl);
     $options = array(
       CURLOPT_RETURNTRANSFER => true,
       CURLOPT_HTTPHEADER => array
         ('Content-type: application/json')
     );
   ```

```
curl_setopt_array( $ch, $options);
$titles = json_decode(curl_exec($ch));
curl_close($ch);
if ((!is_array($titles)) || (count($titles) == 0)) {
  $this->errorAndEnd('create', 'No titles found
    fetching from URL: ' . $jsonUrl);
}
for ($count = 0; $count < count($titles); $count++) {
  if (strlen($titles[$count]) < 8) {
    unset($titles[$count]);
  }
}
if (count($titles) < 1) {
  $this->errorAndEnd('create', 'No suitable titles
    found in database.');
}
$titles = array_merge($titles);  //Renumber the array
$this->render('create', array('titles' => $titles));
}
```

3. Now open `protected/views/hangman/create.php` and make it look like the following listing:

```
<?php
$this->breadcrumbs=array(
  'Hangman'=>array('/hangman'),
 'Create',
);
if (isset($error)) {
  echo("ERROR: $error <br />\n");
}
else {
  echo(print_r($titles, 1) . "\n");
}
```

4. If all has gone well, you should see the dump of the `$titles` array when you go to `http://localhost/cbdb/index.php/hangman/create:http://localhost/cbdb/index.php/hangman/create`.
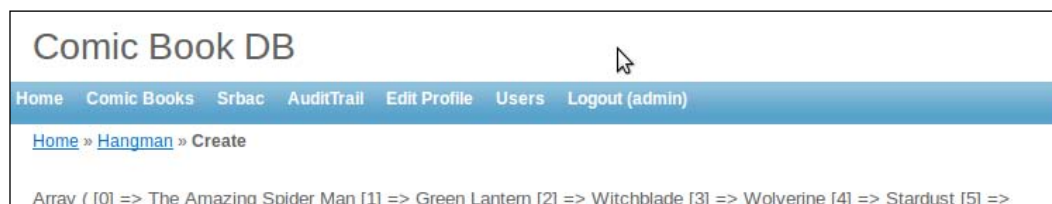
Now we have verified that we are fetching the list of titles with `curl`. If this is not working, you may need to install `curl` or configure PHP.

5. Now, we will look at token creation. Add another function to the `HangmanController` class as follows:

```
private function hangmanToken() {
  $charset = '0123456789abcdef';
  $token = '';
  $charArr = preg_split('//', $charset , 0,
    PREG_SPLIT_NO_EMPTY);
  for ($count = 0; $count < 32; $count++) {
    $token .= $charArr[mt_rand(0, count($charArr) - 1)];
  }
  return $token;
}
```

This will create a 32-digit hexadecimal string representing a 128-bit hexadecimal number (the amount of entropy for the token is a maximum of 128 bits). We will save this string, along with our selected title, and then redirect to the `play` action.

6. Change the lines for `actionCreate()` as indicated.

```
public function actionCreate()
{
  $error = '';
  $request = Yii::app()->request;
  $jsonUrl = $request->hostInfo . $request->baseUrl .
    '/index.php/book/titlelist';
  $ch = curl_init($jsonUrl);
  $options = array(
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => array
      ('Content-type: application/json')
  );
  curl_setopt_array( $ch, $options);
  $titles = json_decode(curl_exec($ch));
  curl_close($ch);
  if ((!is_array($titles)) || (count($titles) == 0)) {
    $this->errorAndEnd('create', 'No titles found
      fetching from URL: ' . $jsonUrl);
  }
  for ($count = 0; $count < count($titles); $count++) {
    if (strlen($titles[$count]) < 8) {
      unset($titles[$count]);
    }
  }
```

217

```
      if (count($titles) < 1) {
        $this->errorAndEnd('create', 'No suitable titles
          found in database.');
      }
      $titles = array_merge($titles);  //Renumber the array


      $hangman = new Hangman;
      $randCount = 0;
      $hangman->title = strtoupper($titles[mt_rand
        (0, count($titles) - 1)]);
      do {
        if ($randCount > 5) {  //Even one duplicate is *highly*
          unlikey (1 in 2^128 if mt_rand were truly random)
          $this->errorAndEnd('create', 'Token generation appears to
            be broken.');
        }
        $hangman->token = $this->hangmanToken();
        $randCount++;
      } while ((Hangman::model()->find('token=:token',
        array(':token'=>$hangman->token))) != null);
      $hangman->save();
      $this->redirect($request->hostInfo . $request->baseUrl .
        '/index.php/hangman/play?token=' . $hangman->token);
    }
```

7. Now redo the view.

```
<?php
$this->breadcrumbs=array(
  'Hangman'=>array('/hangman'),
  'Create',
);?>
<h1>Hangman Game Start Error</h1>

<pre>
<?php echo("ERROR: $error <br />\n"); ?>
</pre>
```

When you go to `http://localhost/cbdb/hangman/create` it should redirect to something like `http://localhost/cbdb/index.php/hangman/play?token=62f36f5c03237af6f94ae952d5a43` (the token will be different, of course):
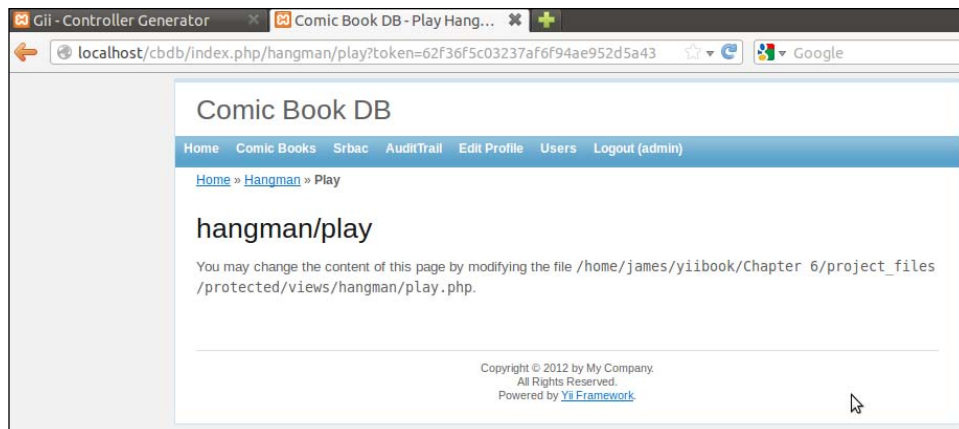
You can also look in the database table and see that a record was created with the expected values.
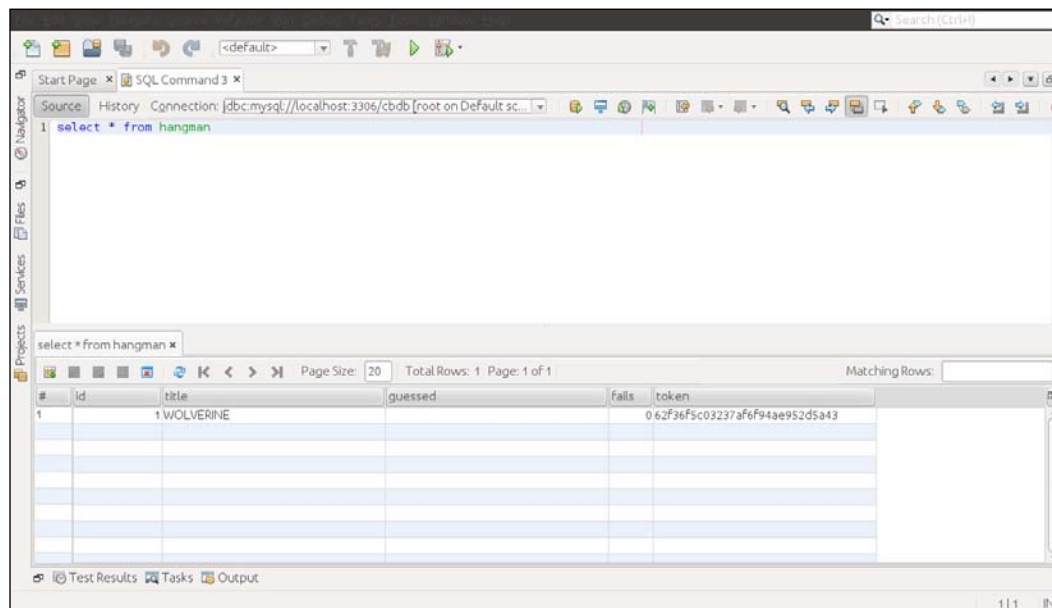
Objective Complete - Mini Debriefing

At the very beginning, we added a function named `errorAndEnd()`. This function simply calls `render()` with an error string and then terminates the application. Then, in the first part of this task, we used `curl` to fetch JSON from the endpoint we created in the previous task and then tested it. When we set `CURLOPT_RETURNTRANSFER` to `true`, this tells `curl_exec()` to return the content of the endpoint as a string. Then, we check to see if any titles were returned, and if none were, we display an error and exit with `errorAndEnd()`. Next, we loop through the list of returned titles and delete any shorter than eight characters (it seems like hangman is not as fun with very short titles). We check to see that we still have some viable titles, once again returning an appropriate error and ending if our check fails. We renumber the array with `array_merge()`, because when you use unset to remove items from an ordinal array in PHP, it doesn't renumber them. The function `array_merge()` can be used as a clever way to fix this. Finally, we call `render()`, passing the newly renumbered array `$titles` to the view.

We create a tiny view that checks for an error. If an error is present, it is displayed, otherwise the list of titles is dumped for debugging purposes, using `print_r()`. This allows us to debug the first part of the task.

Moving on… we needed a token, so we added the function `hangmanToken()`. This function is written simply in order to convey the concept of a unique token in the simplest way possible. However, it is not the best way to generate a token. You will find more information about token generation in the *Classified Intel* section of this task.

With the statement `$hangman->title = strtoupper($titles[mt_rand(0, count($titles) - 1)]);` we select a random title from our list to be uppercased. We then generate a random token and check to make sure it doesn't exist by querying the database. If it does exist, which is extremely unlikely, we create another random token (if we wind up creating an existing token more than five times, something has gone very wrong and we abort). After all this, we add the token to the query string for the URL to `play`, and redirect there.

Classified Intel

When we discuss random numbers in computer science, we are almost always discussing pseudorandom numbers. Computers are deterministic machines by nature, and it is impossible to generate truly random numbers without expensive and highly specialized hardware. When we talk about the **strength** of (pseudo) random numbers, we are actually referring to the predictability of those numbers. While `mt_rand()` generates stronger random numbers than most implementations of `rand()`, it is not the best way to generate strong random numbers suitable for non-guessable tokens and cryptography. A comprehensive discussion of strong random numbers is outside the scope of this book. However, an understanding of this concept is essential to developing secure applications.

The placeholder function we wrote (`hangmanToken()`) is sufficient to make a fun and playable game, but the generated tokens may not withstand the scrutiny of a major government, hacker, or cryptographer who is determined to predict their values. If you wish to implement a more secure version of `hangmanToken()`, we encourage you to do so (see the *You Ready to go Gung Ho? A Hotshot Challenge* section at the end of this chapter).

Developing the Controller – Making the Rules

We will write the play action in the Hangman controller now. This is where the main part of our game will be implemented. We have to take the rules of the game, and figure out how to track them in the database table in a stateful fashion. (Obviously, we already have a good idea of how we're going to do this, since we've already created our database table and discussed what each column is for.) Then, we have to use Yii to update these states and translate them into meaningful interaction for the user.

Engage Thrusters

1. Fetch the row from the database using the token that was passed from the create action (or from the play view we will develop later). If no record is found, display an error and terminate. Add the following lines to the beginning of `actionPlay()` in `protected/controllers/HangmanController.php`:

```
$hangman = Hangman::model()->find('token=:token',
  array(':token'=>$_GET['token']));
if ($hangman == null) {
  $this->errorAndEnd('play', 'Invalid token.');
}
```

2. Change `$this->render('play');` to `$this->render('play', array('token' => $hangman->token));` at the bottom of the function, so we can see what's going on. Now change the view (`protected/views/hangman/play.php`) to reflect this and let's test these small changes:

```
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );
  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
```

221

```
else {
  echo("<h1><center><u>CBDB
    Hangman</u></center></h1><br />\n");
  echo("<h2><center>Token: $token</center></h2>
    <br />\n");

}
```

Now you should see something like the following screenshot:
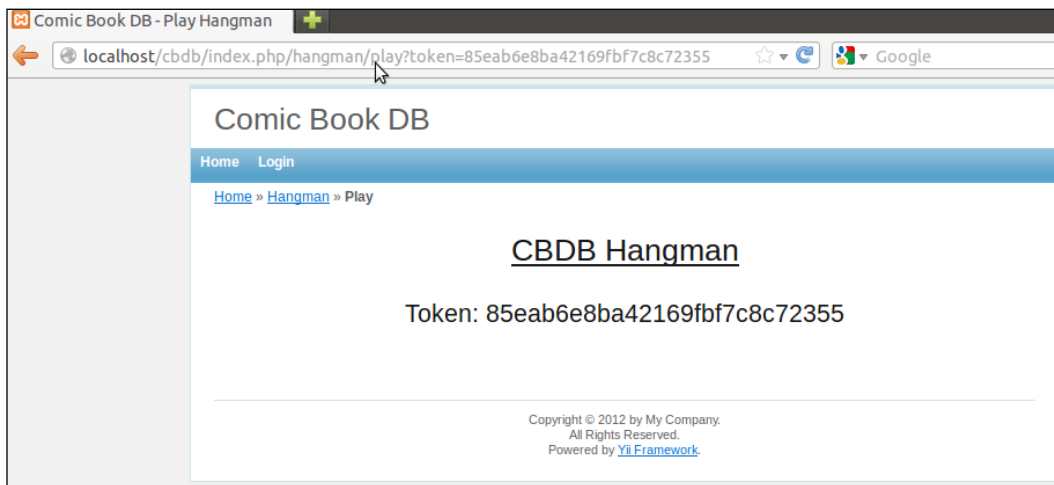
3. Put in an invalid token value and you should see the following screenshot:
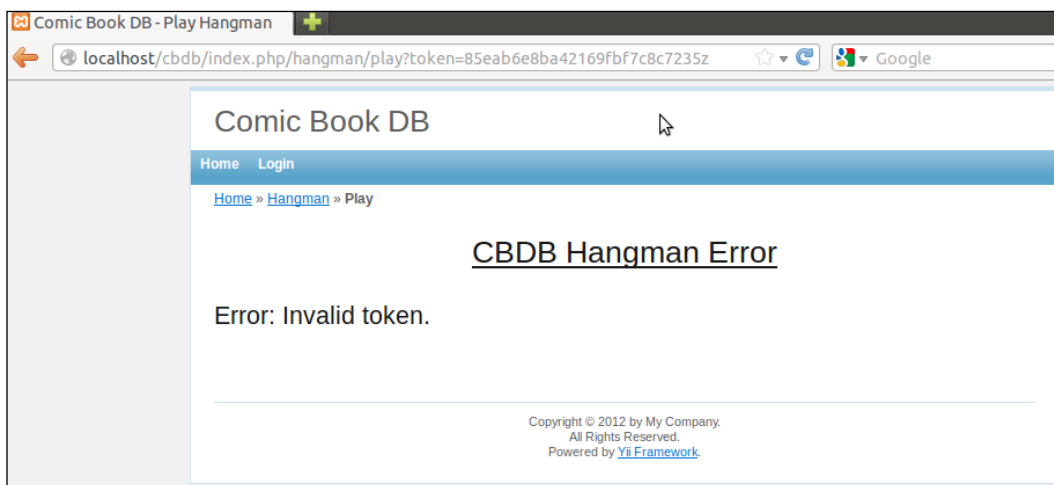
4. Add some additional code to `actionPlay()` and change the render to include the new information.

```php
public function actionPlay()
{
  $hangman = Hangman::model()->find('token=:token',
    array(':token'=>$_GET['token']));
  if ($hangman == null) {
    $this->errorAndEnd('play', 'Invalid token.');
  }
  $title = strtoupper($hangman->title);
  $guessed = array();
  foreach (preg_split('//', $hangman->guessed , 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    $guessed[$letter] = 1;
  }
  $maskedTitle = '';
  foreach (preg_split('//', $title, 0, PREG_SPLIT_NO_EMPTY) as
    $letter) {
    if (!isset($guessed[$letter]) && ctype_alpha($letter)) {
      $maskedTitle .= '_ ';
    }
    else {
      $maskedTitle .= $letter . ' ';
    }
  }
  $maskedTitle = preg_replace('/ /', ' ', $maskedTitle);
  $this->render('play', array('maskedTitle' => $maskedTitle,
    'guessed' => $hangman->guessed));
}
```

5. Once again, change the view so we can see what's going on:

```php
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );
  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
  else {
    echo("<h1><center><u>CBDB
      Hangman</u></center></h1><br />\n");
```

223

```
echo("<h2><center>Title: $maskedTitle</center></h2>
    <br />\n");
echo("<h2><center>Guessed: $guessed</center></h2><br />\n");


    }
?>
```

6. Now creating a new game by going to `http://localhost/cbdb/index.php/hangman/create` or loading an existing game from play should show something like the following screenshot:

7. Now we need to make a way for the controller to check for wins, losses, and to process guesses. We will add a function and change `actionPlay()` to accomplish this:

```
private function assessWin($guesses, $title) {
  $guessArr = array();
  foreach (preg_split('//', strtoupper($guesses), 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    $guessArr[$letter] = true;
  }
  foreach (preg_split('//', strtoupper($title), 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    if (!isset($guessArr[$letter]) && ctype_alpha($letter)) {
      return false;
    }
  }
```

```
    return true;
  }

public function actionPlay()
{
  $message = '';
  if (!isset($_GET['token'])) {
    $this->errorAndEnd('play', 'No token set.');
  }
  $token = $_GET['token'];
  $hangman = Hangman::model()->find('token=:token',
    array(':token'=>$token));
  if ($hangman == null) {
    $this->errorAndEnd('play', 'Invalid token.');
  }

  $title = strtoupper($hangman->title);
  $win = false;
  $lose = false;
  if ($hangman->fails > 5) {
    $lose = true;
  }
  else {
    $win = $this->assessWin($hangman->guessed, $hangman-
      >title);
    if (!$win && isset($_GET['guess'])) {
      $guess = strtoupper($_GET['guess']);
      if (strlen($guess) == 1 && ctype_alpha($guess) &&
        !strstr($hangman->guessed, $guess))  {
        if (!strstr($title, $guess)) {
          $hangman->fails++;
          if ($hangman->fails > 5) {
            $lose = true;
          }
        }
        $hangman->guessed .= $guess;
        $guessed = preg_split('//', $hangman->guessed, 0,
          PREG_SPLIT_NO_EMPTY);
        sort($guessed);
        $hangman->guessed = implode($guessed);
        $hangman->save();
        $win = $this->assessWin($hangman->guessed, $hangman-
          >title);
      }
      else {
```

225

```
        $message .= 'Invalid guess. Please enter a single
          letter that hasn't already been guessed.';
      }

    }
  }
  $guessed = array();
  foreach (preg_split('//', $hangman->guessed, 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    $guessed[$letter] = 1;
  }
  $maskedTitle = '';
  foreach (preg_split('//', $title, 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    if (!isset($guessed[$letter]) &&
      ctype_alpha($letter)) {
      $maskedTitle .= '_ ';
    }
    else {
      $maskedTitle .= $letter . ' ';
    }
  }
  $maskedTitle = preg_replace('/ /', ' ',
    $maskedTitle);

  $this->render('play', array('maskedTitle' => $maskedTitle,
    'guessed' => $hangman->guessed, 'fails' => $hangman->fails,
    'win' => $win, 'lose' => $lose, 'title' => $title,
    'token' => $hangman->token, 'message' => $message));
}
```

8. Make one final change to the view, so we can see what we have done.

```
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );
  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
  else {
    echo("<h1><center><u>Welcome to CBDB
      Hangman</u></center></h1><br />\n");
```

226

```
echo("Title: $maskedTitle<br /><br />\n");
echo("Guessed: $guessed<br />\n");
echo("Fails: $fails<br />\n");
if ($win) {
  echo("<br /><br /><center><h1>You Win!!!</h1></center>
    <br />\n");
}
elseif ($lose) {
  echo("<br /><br /><center><h1>You Lose :(</h1></center>
    <br />\n");
  echo("<h2>The answer was $title.</h2><br />\n");
}

}
```

Now, you can actually play Hangman by appending guesses to the parameter string like this:

```
http://localhost/cbdb/index.php/hangman/play?token=85eab6e8ba42169fbf
7c8c72355&guess=r
```

You will see something like the following screenshot as you guess letters:
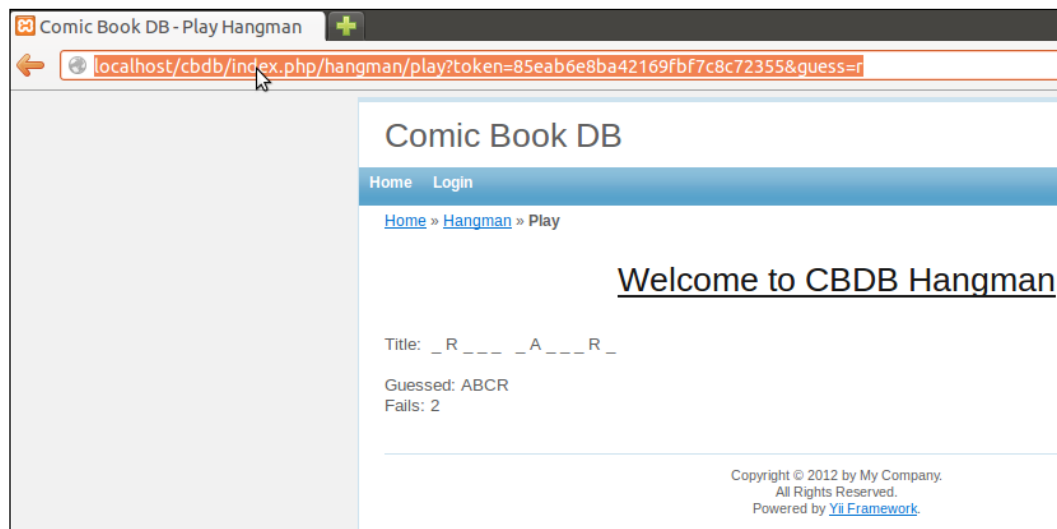
If you repeat a guess, it has absolutely no effect. If you pass more than one letter, or a non-alphabetical character, the guess is not processed. Regardless of whether you enter a lowercase letter, or uppercase letter, the letter is uppercased and processed. If you guess a letter that is not in the title, `$fails` increments (which is displayed in the view). you can see that this one is **GREEN LANTERN**, so you can finish the title and make sure that a win is detected:
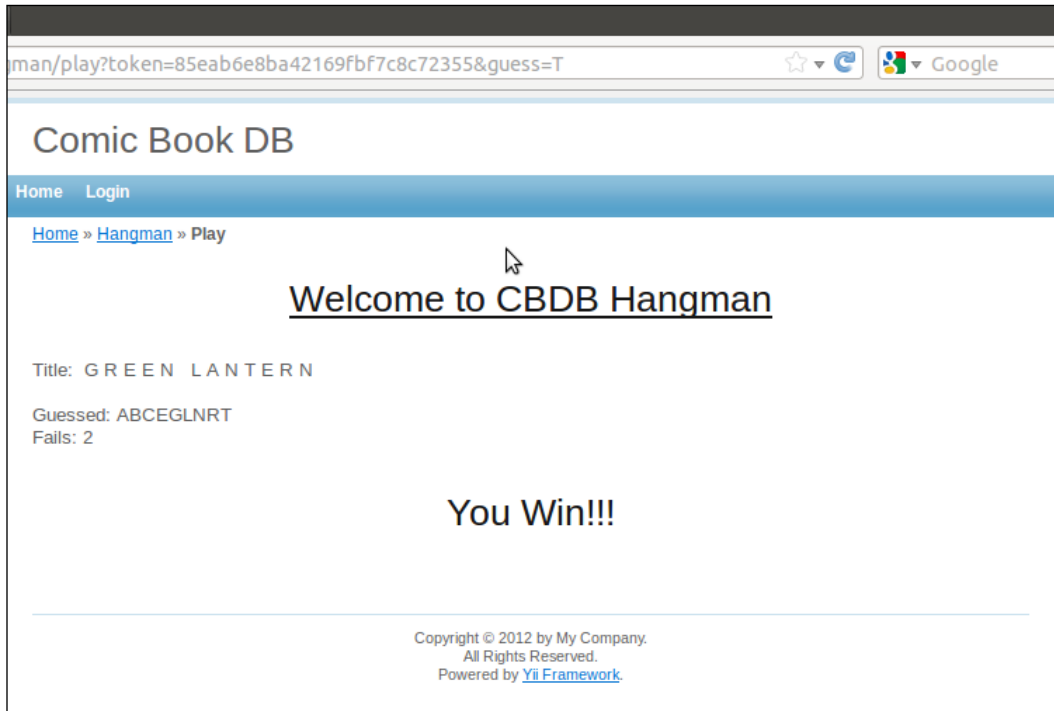
Let's start a new game to test losing. Once you get six failures, a loss is detected. Six failures to a loss is based on these body parts for the hangman: head, torso, left arm, right arm, left leg, and right leg. When a loss is detected, this is what happens:
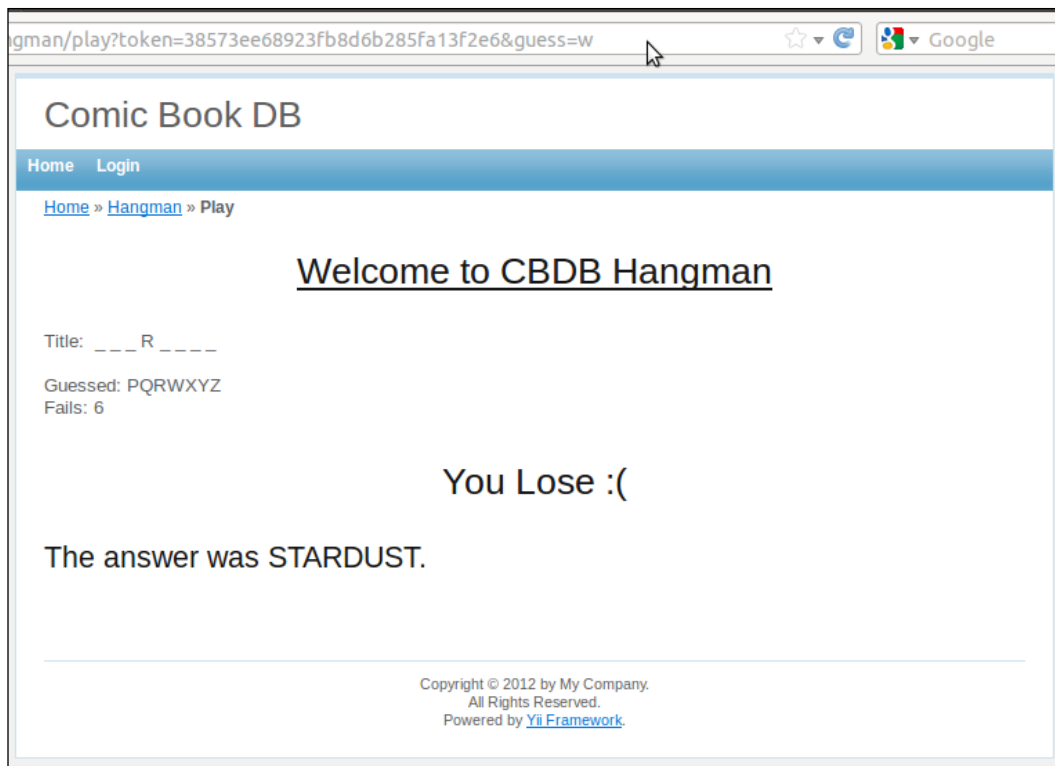
If a particular game is played to a loss or a win, you can see the final state of the game by going to the play URL with the token, but you cannot change it.

Objective Complete - Mini Debriefing

We covered a lot of ground in this task. We are going to go over the final version of the controller and explain each piece of what we have accomplished.

The function `assessWin()` determines if a game has been won by checking to see if all the letters have been guessed, and returns `true` for a win or `false` otherwise. We added the following code snippet to handle the case where the token is not passed at all.

```
if (!isset($_GET['token'])) {
  $this->errorAndEnd('play', 'No token set.');
}
```

Next, logic was added to check for a loss, check for a win, process a guess, and then check for a win again.

When we process the guess, we validate it to make sure it is correct and hasn't already been guessed.

```
if (!$win && isset($_GET['guess'])) {
  $guess = strtoupper($_GET['guess']);
  if (strlen($guess) == 1 && ctype_alpha($guess) &&
    !strstr($hangman->guessed, $guess))  {
    if (!strstr($title, $guess)) {
      $hangman->fails++;
      if ($hangman->fails > 5) {
        $lose = true;
      }
    }
    $hangman->guessed .= $guess;
    $guessed = preg_split('//', $hangman->guessed, 0,
      PREG_SPLIT_NO_EMPTY);
    sort($guessed);
    $hangman->guessed = implode($guessed);
    $hangman->save();
    $win = $this->assessWin($hangman->guessed, $hangman->title);
  }
  else {
    $message .= 'Invalid guess. Please enter a single letter
      that hasn't already been guessed.';
  }

}
```

If the guess is not in the title, we increment `$hangman->fails` and check for a loss. We append the guess to `$guessed`, sort the letters in `$guessed` alphabetically, and reassign the string to `$hangman->guessed`. Then we call `$hangman->save()` to save `$hangman->guessed` (and `$hangman->fails` if needed). Then, we check for a win one final time.

The final call to `$this->render()` is modified to return the values the view will need. The view checks for an error, and if there is no error, displays relevant info about the game.

Developing the View

We've coded all the rules in the play and create actions in our controller, but our view doesn't really look like hangman. Also, guessing the letter by modifying the query string isn't really the most intuitive of user interfaces. However, at this point, we have tested all the functionality of the controller, and it seems solid, so we have a good foundation to start work on the final layer. Let's get to work on the view!

Prepare for Lift Off

We have produced some artwork to help us with this step. The images can be found in the `images/hangman` directory of our `webroot`. If you want, you can take a look at them before we get started, so you can see how this will all fit together.

Engage Thrusters

All the work done in this task will be in the view for play.

1. Start with `protected/views/play.php` as follows:

```php
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );

  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
  else {
    echo("<h1><center><u>Welcome to CBDB
      Hangman</u></center></h1><br />\n");
    echo("<p>\n");
    echo("Title: $maskedTitle<br /><br />\n");
    echo("Fails: $fails<br />\n");
    echo("Guessed: $guessed<br />\n");
    if ($win) {
      echo("<br /><br /><center><h1>You
        Win!!!</h1></center><br />\n");
    }
    elseif ($lose) {
      echo("<br /><br /><center><h1>You Lose
        :(</h1></center><br />\n");
      echo("<h2>The answer was $title.</h2><br />\n");
    }
    echo("</p>\n");
  }
?>
```
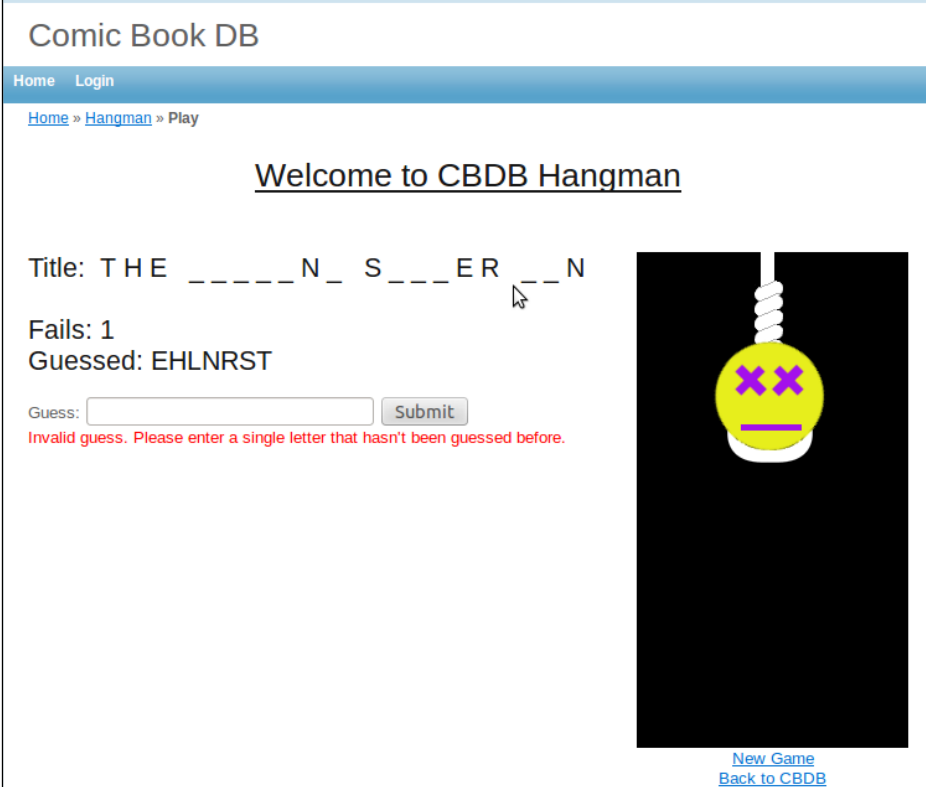
2. Now let's incorporate those images I was showing you, to get started. Below the title **Welcome to CBDB Hangman**, place the following line:

```
echo("<h1><center><u>Welcome to CBDB Hangman</u></center></h1><br
/>\n");
echo("<img src='" . Yii::app()->request->baseUrl .
  "/images/hangman/hangman" . $fails . ".png'/><br />\n");
echo("<p>\n");
echo("Title: $maskedTitle<br /><br />\n");
```

Let's look at what that one line got us:
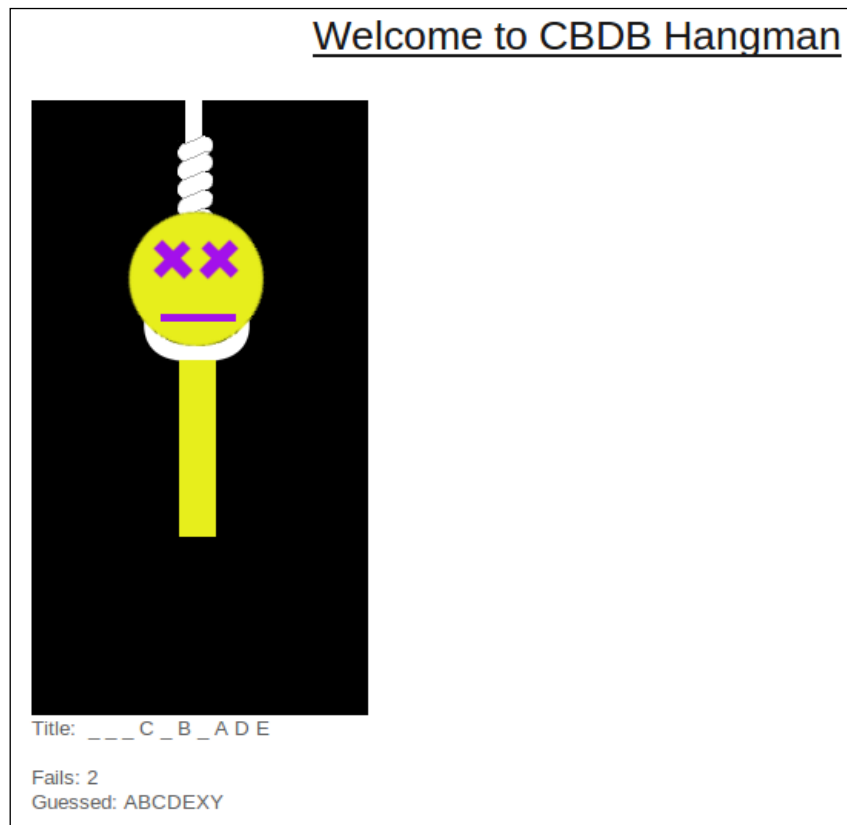
3. It would be nicer if we could move the image to the right and display the interactive text to the left.

```php
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );

  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
  else {
?>
<style type="text/css">
. floatright {
  float: right;
  margin: 0 0 10px 10px;
}
</style>
<h1><center><u>Welcome to CBDB Hangman</u></center></h1><br />
<p>
<?php
    echo("<div class='floatright'>\n");
    echo("<img src='" . Yii::app()->request->baseUrl .
      "/images/hangman/hangman" . $fails . ".png'/>
     <br />\n");
    echo("</div>\n");

    echo("Title: $maskedTitle<br /><br />\n");
    echo("Fails: $fails<br />\n");
    echo("Guessed: $guessed<br />\n");
    if ($win) {
      echo("<br /><br /><center><h1>You
        Win!!!</h1></center><br />\n");
    }
    elseif ($lose) {
      echo("<br /><br /><center><h1>You Lose
        :(</h1></center><br />\n");
      echo("<h2>The answer was $title.</h2><br />\n");
    }
    echo("</p>\n");
  }
?>
```

233

4. Let's make it easy to start a new game, and make it easy to go to the index page. Add the following two lines after the image tag and then throw in some `<h2>` tags:

```
echo("<div class='floatright'>\n");
echo("<img src='" . Yii::app()->request->baseUrl .
  "/images/hangman/hangman" . $fails . ".png'/>
  <br />\n");
echo("<center><a href='" . Yii::app()->request->baseUrl .
  "/index.php/hangman/create'>New Game</a><br />");
echo("<a href='" . Yii::app()->request->baseUrl .
  "/index.php'>Back to CBDB</a><br /></center>");
echo("</div>\n");

echo("<h2>\n");
echo("Title: $maskedTitle<br /><br />\n");
echo("Fails: $fails<br />\n");
echo("Guessed: $guessed<br />\n");
echo("</h2>\n");
```

Now it should look like the following screenshot:
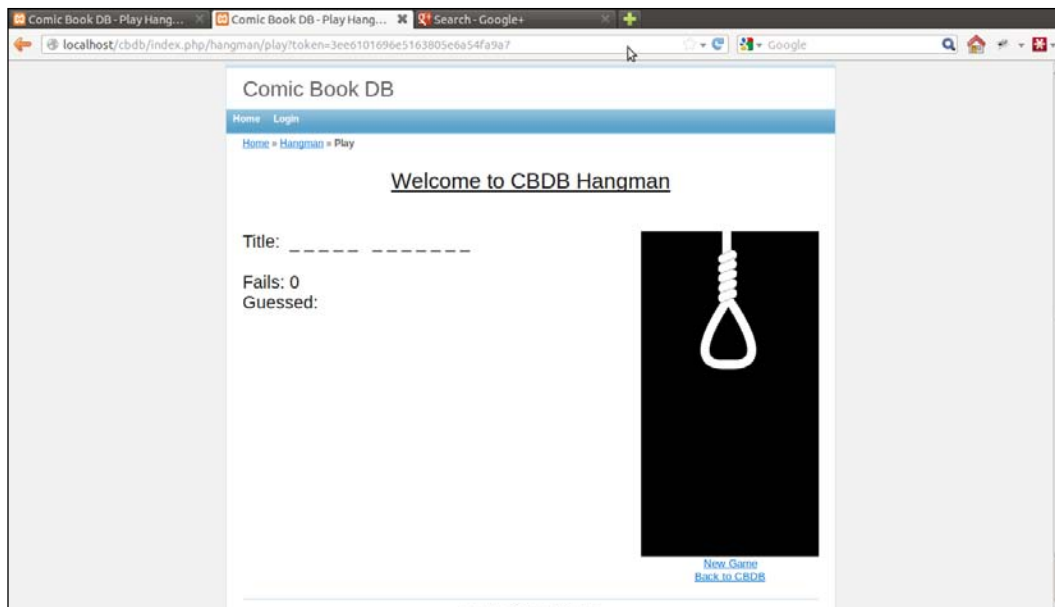
This gives us almost everything we need, with just a few lines of code. The only thing we still need is a way to make guesses, other than manually modifying parameters in the query string. It should be fairly straightforward to make a tiny form that simply submits the guess.

5. Make the final version of your view look like the following code snippet (We moved some things around to make it prettier, and added the aforementioned form), and we'll discuss what we've done:

```php
<?php
  $this->breadcrumbs=array(
    'Hangman'=>array('/hangman'),
    'Play',
  );

  if (isset($error)) {
    echo("<h1><center><u>CBDB Hangman
      Error</u></center></h1><br />\n");
    echo("<h2>Error: $error</h2><br />\n");
  }
  else {
?>
<style type="text/css">
. floatright {
  float: right;
  margin: 0 0 10px 10px;
}

.errormessage {
  color: red;
}
</style>
<h1><center><u>Welcome to CBDB
  Hangman</u></center></h1><br />
<p>
<?php
    echo("<div class='floatright'>\n");
    echo("<img src='" . Yii::app()->request->baseUrl .
      "/images/hangman/hangman" . $fails . ".png'/>
      <br />\n");
    echo("<center><a href='" . Yii::app()->
      request->baseUrl . "/index.php/hangman/create'>New
      Game</a><br />");
    echo("<a href='" . Yii::app()->request->baseUrl .
      "/index.php'>Back to CBDB</a><br /></center>");
    echo("</div>\n");

    echo("<h2>\n");
    echo("Title: $maskedTitle<br /><br />\n");
    echo("Fails: $fails<br />\n");
    echo("Guessed: $guessed<br />\n");
    echo("</h2>\n");
    if ($win) {
      echo("<br /><br /><center><h1>You
        Win!!!</h1></center><br />\n");
```

235

```
      }
      elseif ($lose) {
        echo("<br /><br /><center><h1>You Lose
          :(</h1></center><br />\n");
        echo("<h2>The answer was $title.</h2><br />\n");
      }
      else {
?>
        <form name="guess_form" method="get">
        <?php echo("<input type='hidden' name='token'
         alue='$token'>\n"); ?>
        Guess: <input type="text" name="guess" />
        <input type="submit" value="Submit" />
        </form>
<?php
        echo("<div class='errormessage'>\n");
        echo("$message<br />\n");
        echo("</div>\n");
      }
    }
?>
```

This leaves us here:
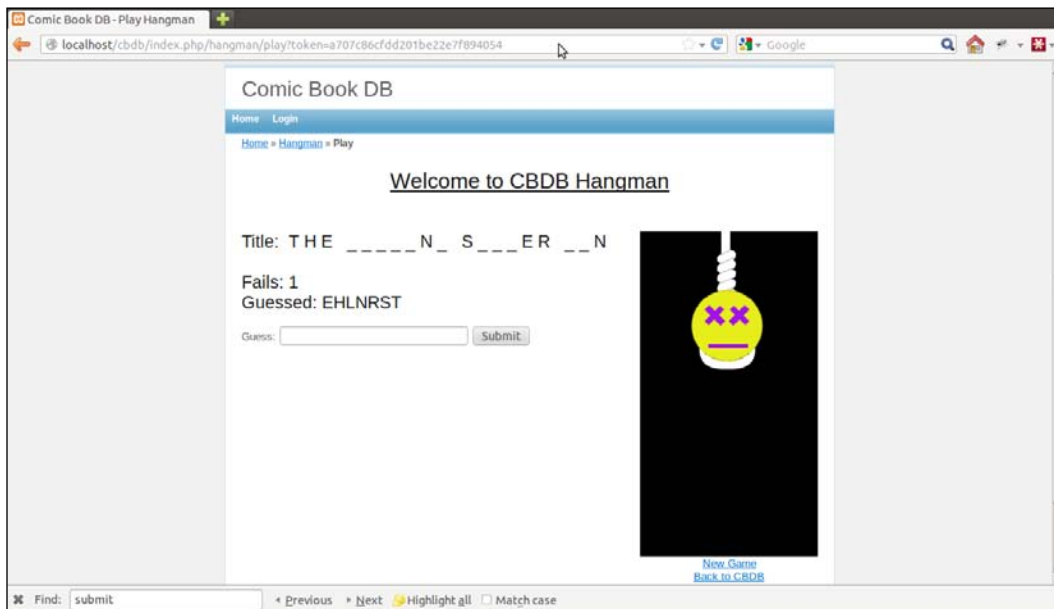
If we put in an invalid guess, this is what happens:

Objective Complete - Mini Debriefing

What we did in this task is all pretty straightforward.

We created some divs and used some style tricks to make it all prettier. Ultimately, our style changes should be placed in `css` files so it is easier to include and maintain them.

When we added the form that submits the guess, we went ahead and added a token as a hidden input field, built to have the value of `$token`, so it will resubmit the token value with the guess. Because of our conditional logic, the form will not be displayed if there is an error, or if the game has already been won or lost.

This works pretty well, but we could make it even better without too much effort. In the next task, we will look at how we can improve the view.

Improving the View

There are a few mildly annoying problems with the game as we have currently implemented it. You have to click on the **Guess** input box to type, and then you can click on **Submit** or press *Enter*. It would be nice if we could set the input focus to the **Guess** input box. It would also be nice if we didn't have to press *Enter*, since we are only guessing one letter at a time. We can use jQuery to quickly fix this.

Engage Thrusters

Let's give the **Guess** text input focus when the page loads. Register a tiny piece of JavaScript to be run on `POS_READY`, and then give the **Guess** input an ID to refer to it from the JavaScript. We will use jQuery to do this because it is convenient and concise (open `protected/views/hangman/play.php`):

```
    else {
      Yii::app()->clientScript->registerScript('guessfocus',"
        $('#guess_id').focus();
      ",CClientScript::POS_READY);
  ?>

      <form name="guess_form" method="get">
        <?php echo("<input type='hidden' name='token'
          value='$token'>\n"); ?>
        Guess: <input type="text" name="guess" id='guess_id'/>
        <input type="submit" value="Submit" />
      </form>
      <?php
        echo("<div class='errormessage'>\n");
        echo("$message<br />\n");
        echo("</div>\n");
      }
    }
  ?>
```

That was easy! Now when you load the page, the **Guess** textbox has the input focus.

We put the `registerScript()` statement inside the `else` clause so it will only run if the form is displayed.

Let's submit the form when we press a key. jQuery has a `keyPress()` function for hooking the event, but they warn us that it is not part of their official API and that it may work differently for different values. We will use it and keep the code simple and straightforward to avoid cross-browser issues:

```
else {
  Yii::app()->clientScript->registerScript('guessfocus',"
    $('#guess_id').focus();
    $('#guess_id').keypress(function(event) {
      if ((event.ctrlKey == false) &&
        (event.altKey == false) &&
        (event.metaKey == false)) {

          event.preventDefault();
          $('#guess_id').val(
          String.fromCharCode(event.charCode));
          $('#guess_form_id').submit();


        }
      });
      ",CClientScript::POS_READY);
?>

    <form name="guess_form" id="guess_form_id" method="get">
      <?php echo("<input type='hidden' name='token'
        value='$token'>\n"); ?>
      Guess: <input type="text" name="guess" id='guess_id'/>
      <input type="submit" value="Submit" />
    </form>
<?php
    echo("<div class='errormessage'>\n");
    echo("$message<br />\n");
    echo("</div>\n");
  }
 }
?>
```

Objective Complete - Mini Debriefing

With asynchronous notification events like this, the sequence of events can be somewhat counter-intuitive. It would seem that we don't need the call `preventDefault()` or the call to `val()` to set the value of `guess_id`. If you remove those lines, the textbox will be updated with the character you pressed, but when the form automatically submits in the `keyPress()` function, the guess will not yet have a value. If you set the value for guess but don't call `preventDefault()`, two copies of the character you pressed will appear in the textbox. If you set the value and then return false from the `keyPress()` function, it appears to work as desired in the Firefox browser, but not necessarily in other browsers.

So now it auto-submits. We don't even need that **Submit** button. Just go ahead and delete it from your form, and enjoy playing hangman.

Authorized Entry Only

We have made a fully functional game. It should now be pointed out that we made it with **Srbac** in debug mode, and we should now lock it down so only logged in users can play. We don't want anyone to be able to go to the **create URL** page and arbitrarily create games and consume server resources. We are quickly going to walk through this, but it is covered fully in *Project 4, Level Up! Permission Levels*. You might want to refer back if you have questions.

Prepare for Lift Off

We have been developing in debug mode. Turn off debug mode in the `srbac` array in `protected/config/main.php`.

```
'srbac' => array(
  'userclass'=>'User', //default: User
  'userid'=>'id', //default: userid
  'username'=>'username', //default:username
  'delimeter'=>'@', //default:-
  //'debug'=>false, //default :false
  'pageSize'=>10, // default : 15
  'superUser' =>'Authority', //default: Authorizer
```

Engage Thrusters

As we didn't make any allowance in our JSON-fetching `curl` code to allow for authentication, we will need to allow anyone to get a title list. This means anyone that can see your site can get a list of all titles. In the *You Ready to go Gung HO? A Hotshot Challenge* section, fixing this is one of the challenges. Let's make it so anyone can access `BookTitlelist` right now, also in the `srbac` array in `protected/config/main.php`:

```
'notAuthorizedView'=> 'application.views.srbac.access_denied',
'alwaysAllowed'=>array( 'SiteLogin','SiteLogout','SiteIndex',
  'SiteError', 'BookTitlelist'),
'userActions'=>array('Show','View','List'),
```

Now we will make some changes so that only users that are logged in can play the game (we don't really care which users, we just want to make sure they have a valid username and password). Before we do this, make sure you can't see the game. Go to `http://localhost/cbdb/index.php/hangman/create` while you're not logged in, and it should redirect you to the login screen. Log in as any user, and you should see the following screenshot:

We need to make a role for playing games and add it to all users. Go to the Srbac menu, click on **Managing Auth Items**, and then click on **Autocreate Auth Items**. Click on the lightbulb next to **Hangman**. **Uncheck Create Tasks**, check **Check All**, and then click on **Create**. It should give the following output:

```
Creating operations
'HangmanCreate' created successfully
'HangmanPlay' created successfully
```

Click on **Managing Auth Items**, then add a task named **playGames**. In the **Description** field, you can enter `Allows users to play games`. Next, create a role called `gamer`. Click on **Assign to Users**, then select **Tasks**. Select **playGames** and add the operations **HangmanCreate** and **HangmanPlay**. Then click on **Roles**, select **Games**, and add the task **playGames**. Srbac is somewhat limited, so we'll need to run the following SQL command in our database:

```
insert into auth_item_child VALUES ('wishlistAccess', 'gamer');
```

This puts games at the bottom of the role hierarchy, below `wishlistAccess`.

Now you should be able to get to the game only if you are logged in.

Objective Complete - Mini Debriefing

We turned debug mode off, added `BookTitlelist` to `alwaysAllowed`, created operations for each Hangman action, added them to a `games` role, and added the `games` role as a child of the `wishlistAccess` role.

Classified Intel

Each time a new game is created, a record is created in the `Hangman` table. A creation timestamp could be added, and a job could be scheduled that deletes records that are beyond a certain age. This could free up disk space and delete unnecessary database records. This will only be a serious issue if you have lots of users playing the game, or if someone is conducting an attack against your site by creating games.

Reusing Code – Making a New Game

Code reusability and maintainability are often touted as two of the most important aspects of corporate software development. Object-oriented programming and MVC frameworks have acquired a great deal of popularity due to the ease of reusing and maintaining code that uses these methodologies. We will make a new game, where an author is given, and you pick the comic book they wrote. We will reuse a lot of the code we wrote in this task so far, and so we should be able to quickly cobble together a new game.

Prepare for Lift Off

The `Hangman` database table can be repurposed as a general table for both games, but it is obviously now misnamed and will also need some minor changes to be suitable for both games. Let's create a new suitable table and drop the other table (if we were in a production situation where the data was valuable, you could dump the data, make some changes, and reimport it to the new table). We will also create a `game_type` table to store different kinds of games. We need to run four database commands.

```sql
CREATE TABLE `game_type` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `devname` varchar(20),
  `name` varchar(40),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

INSERT INTO `game_type` VALUES (0,'hangman',
  'Hangman'),(0,'wrote_it','Wrote It');

CREATE TABLE `game` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `target` varchar(80) NOT NULL DEFAULT '',
  `guessed` varchar(26) NOT NULL DEFAULT '',
  `fails` tinyint(3) unsigned DEFAULT '0',
  `token` varchar(64) NOT NULL,
  `game_type_id` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`game_type_id`) REFERENCES `game_type` (`id`),
  UNIQUE (`token`)
) ENGINE=InnoDB;

DROP TABLE `hangman`;
```

Engage Thrusters

Now that we have our newly adjusted and created tables, we have to make the Hangman game work with the changes. We can just generate new models, and then make the changes in our controller. We are going to have one table, one model, and two controllers with a great deal of shared functionality. Let's get started. Generate the models with Gii. Generate a model named `Gametype` for the `game_type` table, and a model named `Game` for the `game` table. Make sure **Build Relations** is checked.

Now it's time to start making changes to our controller. Create a file named `GameController.php` in `protected/components` and move the functions `hangmanToken()` and `errorAndEnd()` from `protected/controllers/HangmanController.php` and then make the changes shown as follows:

```php
<?php

class GameController extends Controller
{
  protected function gameToken() {
    $charset = '0123456789abcdef';
    $token = '';
    $charArr = preg_split('//', $charset, 0, PREG_SPLIT_NO_EMPTY);
    for ($count = 0; $count < 32; $count++) {
      $token .= $charArr[mt_rand(0, count($charArr) - 1)];
    }
    return $token;
  }

  protected function errorAndEnd($action, $error) {
    $this->render($action, array('error' => $error));
    Yii::app()->end();
  }
}
```

Now let's open the original `HangmanController.php`. Update it to reflect the current changes.

```php
<?php

class HangmanController extends GameController
{
  private function assessWin($guesses, $title) {
    $guessArr = array();
    foreach (preg_split('//', strtoupper($guesses), 0,
      PREG_SPLIT_NO_EMPTY) as $letter) {
      $guessArr[$letter] = true;
    }
    foreach (preg_split('//', strtoupper($title), 0,
      PREG_SPLIT_NO_EMPTY) as $letter) {
      if (!isset($guessArr[$letter]) && ctype_alpha($letter)) {
        return false;
      }
    }
    return true;
```

```
  }

  public function actionCreate()
  {
    $error = '';
    $request = Yii::app()->request;
    $jsonUrl = $request->hostInfo . $request->baseUrl .
      '/index.php/book/titlelist';
    $ch = curl_init($jsonUrl);
    $options = array(
      CURLOPT_RETURNTRANSFER => true,
      CURLOPT_HTTPHEADER => array('Content-type:
        application/json')
    );
    curl_setopt_array( $ch, $options);
    $titles = json_decode(curl_exec($ch));
    curl_close($ch);
    if ((!is_array($titles)) || (count($titles) == 0)) {
      $this->errorAndEnd('create', 'No titles found fetching from
        URL: ' . $jsonUrl);
    }
    for ($count = 0; $count < count($titles); $count++) {
      if (strlen($titles[$count]) < 8) {
        unset($titles[$count]);
      }
    }
    if (count($titles) < 1) {
      $this->errorAndEnd('create', 'No suitable titles found in
        database.');
    }
    $titles = array_merge($titles);  //Renumber the array

    $gameType = Gametype::model()->find('devname="hangman"');

    $hangman = new Game;
    $randCount = 0;
    $hangman->target = strtoupper($titles[mt_rand
      (0, count($titles) - 1)]);
    do {
      if ($randCount > 5) {  //Even one duplicate is *highly*
        unlikey (1 in 2^128 if mt_rand were truly random)
        $this->errorAndEnd('create', 'Token generation appears to
          be broken.');
      }
      $hangman->token = $this->gameToken();
```

```
      $randCount++;
    } while ((Game::model()->find('token=:token',
      array(':token'=>$hangman->token))) != null);
    $hangman->game_type_id = $gameType->id;
    $hangman->save();
    $this->redirect($request->hostInfo . $request->baseUrl .
      '/index.php/hangman/play?token=' . $hangman->token);
  }

  public function actionPlay()
  {
    if (!isset($_GET['token'])) {
      $this->errorAndEnd('play', 'No token set.');
    }

    $game_type = Gametype::model()->find('devname="hangman"');

    $token = $_GET['token'];
    $hangman = Game::model()->find('token=:token',
      array(':token'=>$token));
    if (($hangman == null) || ($hangman->game_type_id !=
      $game_type->id)) {
      $this->errorAndEnd('play', 'Invalid token.');
    }
    $message = '';

    $title = strtoupper($hangman->target);
    $win = false;
    $lose = false;
    if ($hangman->fails > 5) {
      $lose = true;
    }
    else {
      $win = $this->assessWin($hangman->guessed, $hangman-
        >target);
      if (!$win && isset($_GET['guess'])) {
        $guess = strtoupper($_GET['guess']);
        if (strlen($guess) == 1 && ctype_alpha($guess) &&
          !strstr($hangman->guessed, $guess))  {
          if (!strstr($title, $guess)) {
            $hangman->fails++;
            if ($hangman->fails > 5) {
              $lose = true;
            }
          }
```

```
        $hangman->guessed .= $guess;
        $guessed = preg_split('//', $hangman->guessed, 0,
          PREG_SPLIT_NO_EMPTY);
        sort($guessed);
        $hangman->guessed = implode($guessed);
        $hangman->save();
        $win = $this->assessWin($hangman->guessed, $hangman-
          >target);
      }
      else {
        $message .= 'Invalid guess. Please enter a single letter ' .
          'that hasn\'t been guessed before.';
      }

    }
  }
  $guessed = array();
  foreach (preg_split('//', $hangman->guessed, 0,
    PREG_SPLIT_NO_EMPTY) as $letter) {
    $guessed[$letter] = 1;
  }
  $maskedTitle = '';
  foreach (preg_split('//', $title, 0, PREG_SPLIT_NO_EMPTY) as
    $letter) {
    if (!isset($guessed[$letter]) && ctype_alpha($letter)) {
      $maskedTitle .= '_ ';
    }
    else {
      $maskedTitle .= $letter . ' ';
    }
  }
  $maskedTitle = preg_replace('/ /', ' ', $maskedTitle);

  $this->render('play', array('maskedTitle' => $maskedTitle,
    'guessed' => $hangman->guessed, 'fails' => $hangman->fails,
    'win' => $win, 'lose' => $lose, 'title' => $title,
    'token' => $hangman->token, 'message' => $message));
  }
}
```

At this point, Hangman should continue working the way it always has. We now have a `GameController` class that we can extend for additional games.

Let's discuss how our new game "Wrote It" will work. We will select an author and a book written by that author. We will then present the author, and the correct book intermixed with other books the author did not write in a dropdown. If the user picks the proper choice from the dropdown, they win the round. Otherwise, they lose.

1. We need to find the common functionality this game will share with Hangman, and place that functionality in `GameController`. Move the code to `GameController` from `HangmanController`, generalizing and compartmentalizing it for reuse as you go.

```php
<?php

class GameController extends Controller
{
    protected function gameToken() {
    $charset = '0123456789abcdef';
    $token = '';
    $charArr = preg_split('//', $charset, 0,
      PREG_SPLIT_NO_EMPTY);
    for ($count = 0; $count < 32; $count++) {
      $token .= $charArr[mt_rand(0, count
        ($charArr) - 1)];
    }
    return $token;
    }

    protected function fullGameToken() {
      $randCount = 0;
      do {
        if ($randCount > 5) {  //Even one duplicate is *highly*
          unlikey (1 in 2^128 if mt_rand were truly random)
          $this->errorAndEnd('create', 'Token generation appears to
            be broken.');
        }
        $token = $this->gameToken();
        $randCount++;
      } while ((Game::model()->find('token=:token',
        array(':token'=>$token))) != null);
      return $token;

    }

    protected function evalTokenAndGetGame($gameTypeDevname) {
```

```php
    if (!isset($_GET['token'])) {
      $this->errorAndEnd('play', 'No token set.');
    }

    $gameType = Gametype::model()->find('devname=:devname',
      array('devname' => $gameTypeDevname));

    $token = $_GET['token'];
    $game = Game::model()->find('token=:token',
      array(':token'=>$token));
    if (($game == null) || ($game->game_type_id != $gameType-
      >id)) {
      $this->errorAndEnd('play', 'Invalid token.');
    }
    return $game;
  }

  protected function errorAndEnd($action, $error) {
    $this->render($action, array('error' => $error));
    Yii::app()->end();
  }

  protected function getAllTitles($request) {
    $jsonUrl = $request->hostInfo . $request->baseUrl .
      '/index.php/book/titlelist';
    $ch = curl_init($jsonUrl);
    $options = array(
      CURLOPT_RETURNTRANSFER => true,
      CURLOPT_HTTPHEADER => array('Content-type:
        application/json')
    );
    curl_setopt_array( $ch, $options);
    $titles = json_decode(curl_exec($ch));
    curl_close($ch);
    if ((!is_array($titles)) || (count($titles) == 0)) {
      $this->errorAndEnd('create', 'No titles found
        fetching from URL: ' . $jsonUrl);
    }
    return $titles;
  }
}
```

2. Now change `protected/controllers/HangmanController.php` to reflect the new changes.

```php
<?php

class HangmanController extends GameController
{
  private function assessWin($guesses, $title) {
    $guessArr = array();
    foreach (preg_split('//', strtoupper($guesses), 0,
      PREG_SPLIT_NO_EMPTY) as $letter) {
      $guessArr[$letter] = true;
    }
    foreach (preg_split('//', strtoupper($title), 0,
      PREG_SPLIT_NO_EMPTY) as $letter) {
      if (!isset($guessArr[$letter]) &&
        ctype_alpha($letter)) {
        return false;
      }
    }
    return true;
  }

  public function actionCreate()
  {
    $error = '';
    $request = Yii::app()->request;
    $titles = $this->getAllTitles($request);

    for ($count = 0; $count < count($titles); $count++) {
      if (strlen($titles[$count]) < 8) {
        unset($titles[$count]);
      }
    }
    if (count($titles) < 1) {
      $this->errorAndEnd('create', 'No suitable titles
        found in database.');
    }
    $titles = array_merge($titles);  //Renumber the array

    $game_type = Gametype::model()-
      >find('devname="hangman"');

    $hangman = new Game;
    $hangman->target = strtoupper($titles[mt_rand
      (0, count($titles) - 1)]);
    $hangman->token = $this->fullGameToken();
    $hangman->game_type_id = $game_type->id;
```

```
      $hangman->save();
      $this->redirect($request->hostInfo . $request
        ->baseUrl . '/index.php/hangman/play?token=' .
        $hangman->token);
    }

    public function actionPlay()
    {
      $hangman = $this->evalTokenAndGetGame('hangman');
      $message = '';

      $title = strtoupper($hangman->target);
```

Once again, if all has gone well, the Hangman game should now be working in exactly the same way it always has. The code still remaining in `HangmanController.php` is now all specific to Hangman, and generalized code that Hangman will share with the game we are about to write is now in `GameController.php`.

Let's get to work on our new game. We need to add a few additional fields to our database table now that we know where we are going. We will also need to regenerate the model.

1. Run the following statements for your database:

    ```
    ALTER TABLE `game` ADD COLUMN `book_decoy3_id` int(10) unsigned
    DEFAULT NULL AFTER `guessed`;
    ALTER TABLE `game` ADD COLUMN `book_decoy2_id` int(10) unsigned
    DEFAULT NULL AFTER `guessed`;
    ALTER TABLE `game` ADD COLUMN `book_decoy1_id` int(10) unsigned
    DEFAULT NULL AFTER `guessed`;
    ALTER TABLE `game` ADD COLUMN `author_id` int(10) unsigned DEFAULT
    NULL AFTER `guessed`;
    ALTER TABLE `game` ADD COLUMN `win` BOOLEAN DEFAULT 0;
    ALTER TABLE `game` ADD COLUMN `book_id` int(10) unsigned DEFAULT
    NULL AFTER `guessed`;
    ALTER TABLE `game` ADD FOREIGN KEY (`book_id`) REFERENCES `book`
    (`id`);;
    ALTER TABLE `game` ADD FOREIGN KEY (`author_id`) REFERENCES
    `person` (`id`);
    ALTER TABLE `game` ADD FOREIGN KEY (`book_decoy1_id`) REFERENCES
    `book` (`id`);
    ALTER TABLE `game` ADD FOREIGN KEY (`book_decoy2_id`) REFERENCES
    `book` (`id`);
    ALTER TABLE `game` ADD FOREIGN KEY (`book_decoy3_id`) REFERENCES
    `book` (`id`);
    ```

    Gii won't generate code if the file already exists. Delete the current `Game` model.

    rm protected/models/Game.php

2. Now generate the model with Gii:

```
Generating code using template "/opt/lampp/htdocs/
  yii-1.1.10.r3566/framework/gii/generators/model/
  templates/default"...
 generated models/Game.php
done!
```

3. Go to the Gii Controller Generator and generate a controller named `Wroteit` with a base class of `GameController` and the actions `create`, `play`, and `index`.

```
Generating code using template "/opt/lampp/htdocs/
  yii-1.1.10.r3566/framework/gii/generators/controller/
  templates/default"...
 generated controllers/WroteitController.php
 generated views/wroteit/create.php
 generated views/wroteit/index.php
 generated views/wroteit/play.php
done!
```

4. We need a new relationship in the `Person` model as well (`protected/models/Person.php`) so we can easily determine which books a particular author has written. Add the following relationship:

```php
public function relations()
  {
    // NOTE: you may need to adjust the relation name and
    //the related
    // class name for the relations automatically
    //generated below.
    return array(
      'books' => array(self::MANY_MANY, 'Book',
        'bookauthor(book_id, author_id)',
        'index'=>'id'),
      'bookauthors' => array(self::HAS_MANY,
        'Bookauthor', 'author_id'),
      'bookillustrators' => array(self::HAS_MANY,
        'Bookillustrator', 'illustrator_id'),
    );
  }
```

5. We need to update the controller for `create` and `update`, the way we did for Hangman (`protected/controllers/WroteitController.php`).

```php
<?php

class WroteitController extends GameController
{
  private function selectRandomAuthorWithBook($action) {
```

252

```php
    $bookauthors = BookAuthor::model()->findAll(array(
      'select'=>'author_id',
      'group'=>'author_id',
      'distinct'=>true,
    ));
    $authorIds = array();
    foreach ($bookauthors as $bookauthor) {
      $authorIds[] = $bookauthor['author_id'];
    }
    if (count($authorIds) == 0) {
      $this->errorAndEnd($action, 'No authors in
        database.');
    }
    $author = Person::model()->find('id=:id', array
      ('id'=>$authorIds[mt_rand(0, count($authorIds) -
      1)]));
    $bookIds = array();
    foreach ($author->books as $book) {
      $bookIds[] = $book['id'];
    }
    if (count($bookIds) == 0) {
      $this->errorAndEnd($action, 'Relational integrity
        error.  You should not see this.');
    }
    $bookIndex = mt_rand(0, count($bookIds) - 1);
    return array(
      'author_id' => $author['id'],
      'book_id' => $bookIds[mt_rand(0, count
        ($bookIds) - 1)],
    );
  }

  //This function will return three books that are not
  //written by the author referenced by author_id
  private function selectThreeSuitableBooks
    ($author_id, $action) {
    $author = Person::model()->find
      ('id=:id', array('id'=>$author_id));
    $bookIdsByAuthor = array();
    foreach ($author->books as $book) {
      $bookIdsByAuthor[] = $book['id'];
    }
    $criteria = new CDbCriteria;
    $criteria->addNotInCondition('id', $bookIdsByAuthor);
    $ret = array();
```

253

```
      $books = Book::model()->findAll($criteria);
      $bookIds = array();
      foreach ($books as $book) {
        $bookIds[] = $book['id'];
      }
      if (count($bookIds) < 3) {
        $this->errorAndEnd($action, 'Not enough books not
          written by author in database.');
      }
      elseif (count($bookIds) == 3) {
        return $bookIds;
      }
      else {
        for ($count = 0; $count < 3; $count++) {
          $index = mt_rand(0, count($bookIds) - 1);
          $ret[] = $bookIds[$index];
          unset($bookIds[$index]);
          $bookIds = array_merge($bookIds);
        }
      }
      return $ret;
    }

    public function actionCreate()
    {
      $randbook = $this
        ->selectRandomAuthorWithBook('create');
      $decoyIds = $this->selectThreeSuitableBooks
        ($randbook['author_id'], 'create');
      $gameType = Gametype::model()
        ->find('devname="wrote_it"');
      $wroteIt = new Game;
      $wroteIt->token = $this->fullGameToken();
      $wroteIt->game_type_id = $gameType['id'];
      $wroteIt->book_id = $randbook['book_id'];
      $wroteIt->author_id = $randbook['author_id'];
      $wroteIt->book_decoy1_id = $decoyIds[0];
      $wroteIt->book_decoy2_id = $decoyIds[1];
      $wroteIt->book_decoy3_id = $decoyIds[2];
      $wroteIt->save();
      $request = Yii::app()->request;
```

```
    $this->redirect($request->hostInfo . $request
      ->baseUrl . '/index.php/wroteit/play?token=' .
      $wroteIt->token);
}

public function actionIndex()
{
  $this->render('index');
}

public function actionPlay()
{
  $wroteIt = $this->evalTokenAndGetGame('wrote_it');
  $win = false;
  if (!$wroteIt->win && $wroteIt->fails == 0 &&
    isset($_GET['guess'])) {
    $guess = $_GET['guess'];
    if (strlen($guess) != 0) {
      if ($guess != $wroteIt->book_id) {
        $wroteIt->fails++;
      }
      else {
        $win = true;
        $wroteIt->win = true;
      }
      $wroteIt->save();
    }
  }
  elseif ($wroteIt->win) {
    $win = true;
  }
  $ids = array(
    $wroteIt->book_id, $wroteIt->book_decoy1_id,
    $wroteIt->book_decoy2_id, $wroteIt->book_decoy3_id
  );
  $criteria = new CDbCriteria;
  $criteria->addInCondition('id', $ids);
  $choices = array();
  $author = Person::model()->find('id=:id', array
    ('id' => $wroteIt->author_id));
  $books = Book::model()->findAll($criteria);
  shuffle($books);
  foreach ($books as $book) {
    $choices[] = array('id' => $book->id, 'title' =>
      $book->title);
```

```
    }
    $answer = '';
    $lose = false;
    if ($wroteIt->fails > 0) {
      $lose = true;
    }
    if ($win || $lose) {
      $bookAnswer = Book::model()->find
        ('id=:id', array('id' => $wroteIt->book_id));
      $answer = $bookAnswer->title;
    }

    $this->render('play', array('choices' => $choices,
      'author' => $author['fname'] . ' ' .
        $author['lname'],
      'win' => $win, 'lose' => $lose, 'token' =>
        $wroteIt->token,
      'answer' => $answer)
    );
  }
}
```

6. Make a simple fall-back view for `create` (`protected/views/wroteit/create.php`).

```php
<?php
$this->breadcrumbs=array(
  'Wroteit'=>array('/wroteit'),
  'Create',
);
?>

<h1>WroteIt Game Start Error</h1>
<pre>
<?php if (isset($error)) {echo("ERROR: $error
  <br />\n");} ?>
</pre>
```

7. Now create a view for `play` (`protected/views/wroteit/play.php`).

```php
<?php
  $this->breadcrumbs=array(
    'Wroteit'=>array('/wroteit'),
    'Play',
```

```
    );


    if (isset($error)) {
      echo("<h1><center><u>CBDB WroteIt
        Error</u></center></h1><br />\n");
      echo("<h2>Error: $error</h2><br />\n");
    }
    else { //no $error
?>
<h1><center><u>Welcome to CBDB
  WroteIt</u></center></h1><br />
<?php
    if ($win) {
       echo("<center><h1>You Win!!!</h1><br />\n");
       echo("<h2>$author wrote $answer.</h2></center>
         <br />\n");
    }
    elseif ($lose) {
       echo("<center><h1>You Lose :(</h1><br />\n");
       echo("<h2>$author wrote $answer.</h2></center>
         <br />\n");
    }
    else { //no win or lose
       echo("<center><h2>Author: $author</h2></center>
         <br />");
?>
<form name="guess_form" id="guess_form_id" method="get">
<?php echo("<input name='token' type='hidden'
  value='$token'>\n");?>
<center>
What did this author write?
<select name='guess'>
  <option value="" style="display:none;"></option>
<?php
  foreach($choices as $choice) {
    echo('<option value="' . $choice['id'] . '">');
    echo($choice['title']);
    echo("</option>\n");
  }
?>
</select>
<input type="submit" value="Submit">
</center>
</form>
```

```php
<?php
    } //no win or lose
  } //no $error
  echo("<center><a href='" . Yii::app()->request->baseUrl
    . "/index.php/wroteit/create'>New Game</a><br />");
  echo("<a href='" . Yii::app()->request->baseUrl .
    "/index.php'>Back to CBDB</a><br /></center>");
?>
```

8. We need to set up permissions for users to run the app as well. Go into the Srbac menu as the admin user and auto-create the operations for `Wroteit`. Then, add the operations to the `playGames` role. At this point, we have two fully functional games.

   `Wroteit` should look like the following screenshot:

Objective Complete - Mini Debriefing

We already developed one game, so the contents of our new controller and view should look pretty familiar. When we wrote them, we followed the same order of operations we did for Hangman. We generated the model(s), and then developed and tested the controller with minimal views, and then focused on the views. For the sake of avoiding repetition, we did not walk through every step this time. We mainly wanted to focus on the process of finding common functionality and moving it into a common base class, and then using the base class to make something new.

Mission Accomplished

We made a Hangman game, maintained it with the intention of reusing basic functional components to make a new game, and then made the new game named `WroteIt`.
We did a lot for one chapter. It's pretty fun!

You Ready to go Gung HO? A Hotshot Challenge

On Linux systems, `/dev/random` and `/dev/urandom` are typically the best sources of random numbers. System entropy is typically incorporated into these devices to add entropy to the generated output. The `/dev/random` device is blocking, which means if the system does not contain enough entropy for it to generate the required number of bits, it will wait until enough system entropy is available (based on the general "busyness" of the system) to generate the number. The `/dev/urandom` device will take whatever system entropy is available and generate the remaining number of bits via other pseudorandom means. The challenge is to write your own PHP function that uses one of these devices to generate our token. Experiment with how busy the system has to be to use `/dev/random` rather than `/dev/urandom`. You can also use the function `open_random_pseudo_bytes()` (especially if you need a cross-platform solution). See the PHP documentation available at `http://php.net/manual/en/function.openssl-random-pseudo-bytes.php` for details.

In addition to improving the security of the token, there are other projects we could try. Putting everything in the same database table was a little kludgy. To do it properly, we should have one common game table with the token and perhaps win and lose (the things common between the games) and then we should have two other tables for `Hangman` and `WroteIt` that relate to the `Games` table with a foreign key. The game `WroteIt` could be reworked to be more exciting in a variety of ways, such as serving a large number of rounds to the user and tracking their performance.

259

Project 7

Let It Work While You Sleep – Reports and Job Queues

This project is all about reporting and scheduling. We will create a job queue to schedule resource intensive work for off-peak hours and a report on our data to demonstrate some reporting/presentation packages that you can use with Yii.

Mission Briefing

The idea of writing a job is to manage a bit of code that will consume a noticeable amount of resources to execute. It is something you do not need to execute immediately, so you can load the work, ask it to run at a time where it will least impact your site, and then check on the results later.

One type of process that fits this description is correspondence, and another type is report generation.

We will build a system for managing and scheduling jobs, and use it to schedule an e-mail sending job and a graphical reporting job that produces a chart like the one shown in the following screenshot:
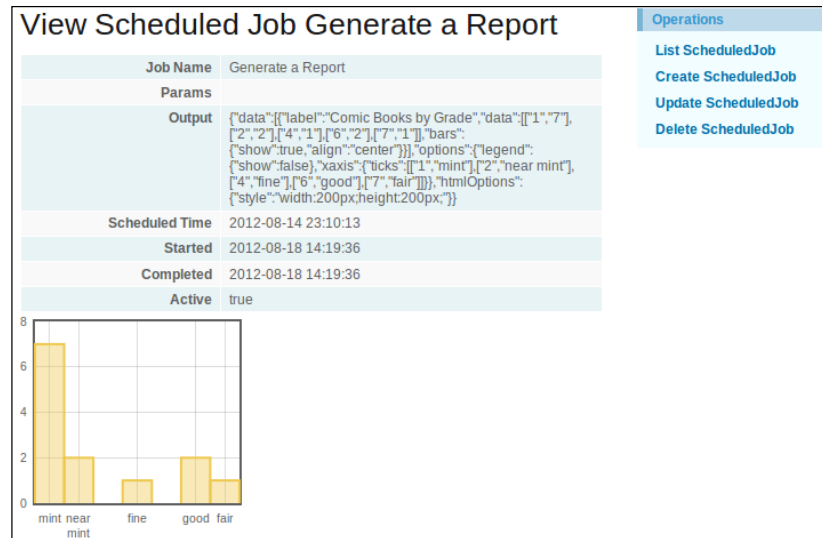
Why Is It Awesome?

As this system is your personal system, you can use it to send e-mails or reports to you or perform any task you want, at any time you want. If you work on commercial systems that serve a large number of users, you cannot perform these actions so flexibly. This chapter will demonstrate some techniques and systems that you can use when you work on highly-available web systems.

Your Hotshot Objectives

In this project, we will cover the following tasks:

- ▶ Reorganizing Menu Items
- ▶ Scaffolding the Job Objects
- ▶ Adding Job Registration
- ▶ Adding Job Scheduling
- ▶ Adding Job Processing
- ▶ Creating and Registering a Job
- ▶ Creating a Graphical Report
- ▶ Displaying Graphical Report Output

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, follow these steps, replacing the username `lomeara` with your own username.

1. Copy the project files into your working directory.

   ```
   cp –r ~/Downloads/project_files/Chapter\ 7/project_files
     ~/projects/ch7
   ```

2. Make the directories that Yii uses web writeable. For example, by using the following command we change ownership of the directories so that our user owns them, but the web group, `www-data`, can read, write, and execute the directories and contents, as well.

   ```
   cd ~/projects/ch7/
   ```

   ```
   sudo chown -R lomeara:www-data protected/runtime assets
     protected/models protected/controllers protected/views
   ```

3. Create a link in the `webroot` directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   ```

   ```
   sudo ln -s ~/projects/ch7 cbdb
   ```

4. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`) and configure for Yii development with PHPUnit.

5. Create a database named `cbdb` and load the database schema (`~/projects/ch7/protected/data/schema.sql`) into it.

6. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans: **ch7** | **Source Files** | **protected** | **config** | **main.php**).

> [Note that the admin login to the web application is admin/test.]

Reorganizing Menu Items

At the start, we have several administrative menu options for our site. We will reorganize them and add the new options for this chapter.

Engage Thrusters

1. Edit **ch7** | **Source Files** | **protected** | **views** | **layouts** | **main.php** and create a new top-level item to collect administrative tasks. Set visible to true, so that the new items render.

```
'items'=>array(
  array('label'=>'Home', 'url'=>array('/site/index'),
    'visible' => true),
  array(
    'label'=>'Comic Books',
    'url'=>array('/book/index'),
    'items' => array(
      array('label'=>'Publishers',
        'url'=>array('/publisher/index')),
      array('label'=>'WishList',
        'url'=>array('/wish/index')),
      array('label'=>'Library',
        'url'=>array('/library/index')),
      ),
      'authItemName' => 'WishlistAccess',
  ),
          array(
            'label'=>'Admin',
            'url' => '',
            'visible' => true,
          ),
```

The menu will now display an **Admin** option.
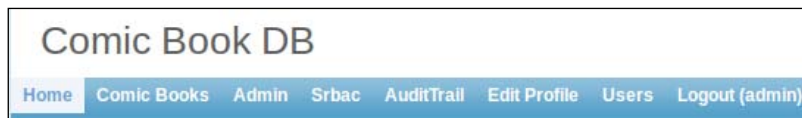
2. Move the existing administrative items srbac, audit trail, and users into the array for this new Admin item:

```
array(
  'label'=>'Admin',
  'url' => '',
  'items' => array(
    array('label'=>'Srbac', 'url'=>array('/srbac'),
      'authItemName' => 'Authority'),
    array('label'=>'AuditTrail', 'url'=>array
      ('/auditTrail/admin'), 'authItemName' => 'Authority'),
    array('label'=>'Users', 'url'=>array('/user/index')),
  ),
  'visible' => true
),
```

The resulting menu will look like the following screenshot:
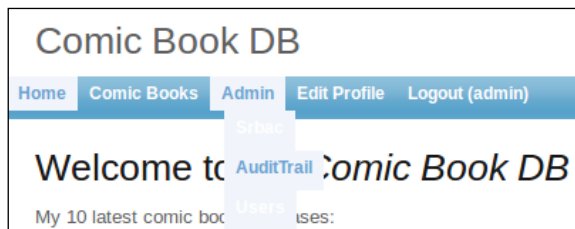
3. Change the value of `visible` to check if access is allowed for any of the administrative menu items. Checking access for the `UserIndex` operation or `Authority` role covers the menu items for now.

```
'visible' => Yii::app()->user->checkAccess('UserIndex') ||
    Yii::app()->user->checkAccess('Authority'),
```

4. Add menu items for the new features that we will be creating later in this project.

```
array('label'=>'Jobs', 'url'=>array
    ('/scheduledJob/index')),
array('label'=>'Reports', 'url'=>array
    ('/report/index')),
```

You will not see these new menu items, because the menu is rendered by YiiSmartMenu. We have not created an operation and assigned authorization for these new items, yet.

5. Create operations `ScheduledJobIndex` and `ReportIndex`. See *Project 5*, *Service Please – Integrating Service Data*, for details on creating operations.
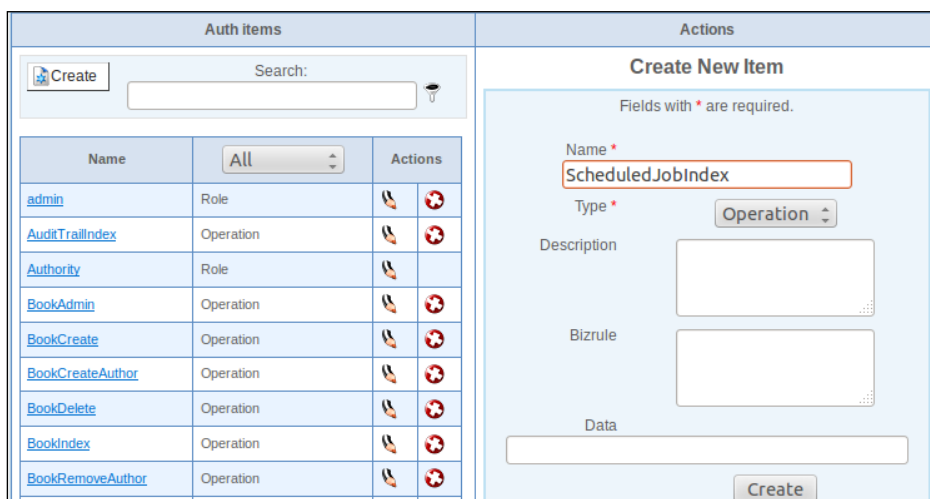
6. Create the task `jobsAndReports`.

7. Assign new operations `ScheduledJobIndex` and `ReportIndex` to the new task `jobsAndReports`.

8. Assign the task `jobsAndReports` to the role `admin`.

9. Now that our admin user is authorized to access the new actions, the new items will appear in the **Admin** drop-down menu.
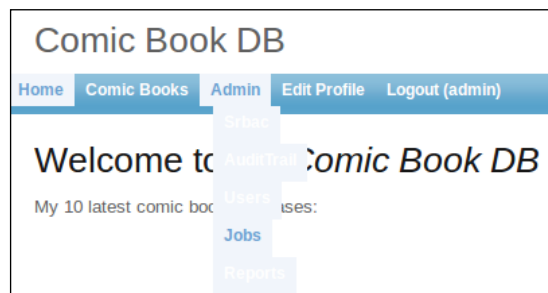
Objective Complete - Mini Debriefing

In this task, we reorganized the site menu to collect all administrative tasks in a single category. We added new items, which we will flesh out in future tasks in this project. Because we are using YiiSmartMenu to render menu items only if the user is authorized to access them, we used the `checkAccess` function to determine whether or not to display the **Admin** menu item, and we had to set up new actions in the role-based access control system before they would render in the menu.

Classified Intel

To be thorough, add the new actions to the visible condition for the menu. This will ensure that if we reorganize the authorization hierarchy in the future, the **Admin** menu will still render for any user authorized to use one of the items in the list.

```
'visible' => Yii::app()->user->checkAccess('UserIndex') ||
  Yii::app()->user->checkAccess('Authority') || Yii::app()-
  >user->checkAccess('ReportIndex') || Yii::app()->user-
  >checkAccess('ScheduledJobIndex') ,
```

Scaffolding the Job Objects

We will use two objects to manage the jobs. One is the job, which holds information about jobs registered with our system. The other is the scheduled job, which is the job queue entry. It represents a request to run a job, and contains the job results and any reported output when the job has finished running. In this section, we will scaffold both objects.

Engage Thrusters

1. We have provided a table definition for the jobs – ready for your use. To load it in NetBeans, open an SQL command window for the `cbdb` database.

2. Right-click on the command window and select **Select in Projects**.

3. Navigate to the jobs SQL file, **ch7 | Source Files | protected | data | job.sql,** in the project window.

4. Copy the contents from `jobs.sql` and paste it into the command window.

5. Hit *Shift + F6* to run the command, and check the SQL output for success.

6. Do the same for the scheduled jobs schema **ch7 | Source Files | protected | data | scheduled_job.sql**.

7. Make sure that the web server can write to the models, views, and controllers directory in your project. In Unix, we use the `chown` command to give write permissions to the `www-data` group.

```
cd ~lomeara/projects/ch7/protected
chown lomeara:www-data models/
chown lomeara:www-data views/
chown lomeara:www-data controllers/
```

8. Use Gii to generate a model and CRUD from the job table and the scheduled job table.

   You can now click on **Jobs** in the **Admin** menu to access the **Jobs Index** list page, but you will not yet see any job actions such as **Create**.

9. In srbac, create operations for the CRUD actions. You already created one for `ScheduledJobIndex` in the previous task. You can go to **Managing auth items | Autocreate Auth Items** to create all of the operations at once. Remember to uncheck the option to create tasks, because we have already created a container task, `jobsAndReports`.

10. In **Srbac | Assign to users | Tasks**, add all of the job operations to the `jobsAndReports` task.
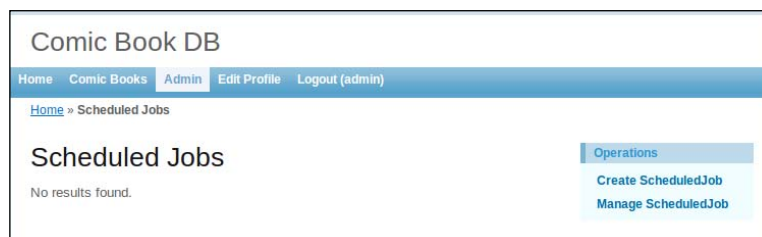
11. Replace `index.php` with `admin.php` in both the `jobs` and `scheduledJobs` view directories.

12. In both `JobController` and `ScheduledJobController`, delete the `actionIndex` and `accessRules` functions. Remove the `accessControl` entry in the filter function. Rename `actionAdmin` to `actionIndex`. Search and replace `admin` with `index`.

13. Now we will create a way to access the job list from the job queue. Edit **ch7 | Source Files | protected | views | scheduledJob | index.php**. Remove `List ScheduledJob` from the array, and add an entry to `List Registered Jobs` and `Register Job`.

```
$this->menu=array(
    array('label'=>'Schedule Job', 'url'=>array('create')),
    array('label'=>'List Registered Jobs',
        'url'=>array('job/index')),
    array('label'=>'Register Job', 'url'=>array('job/create')),
);
```

This will result in the `scheduledJob` index looking similar to the following screenshot:

Objective Complete - Mini Debriefing

In this task, we have created tables to hold job data and our job queue. We used Gii to produce an initial scaffolding, and srbac to create access control entries. Finally, we made some general changes to the scaffolding and created a link in the interface between the two objects.

Adding Job Registration

Before we can add jobs to the queue, we need to be able to register a job with the system. We will build on the job scaffolding in this task to the point where we can manage the jobs registered with our site.

Prepare for Lift Off

1. Create an extensions directory in your project.

   mkdir ~/projects/ch7/protected/extensions

2. Use wget to download the Yii extension quickdlgs from
   http://www.yiiframework.com/extension/quickdlgs/ as follows:

   cd ~/Downloads

   **wget http://www.yiiframework.com/
     extension/quickdlgs/files/quickdlgs.1.2.zip**

   We are going to use the quickdlgs extension to add modal dialogs to create,
   add, and edit entries in our grids.

3. Unzip the package in your project's extensions directory as follows:

   cd ~/projects/ch7/protected/extensions/

   unzip ~/Downloads/quickdlgs.1.2.zip

4. Add the following entry to the import array in **ch7** | **Source Files** | **protected** |
   config | **main.php**.

   'ext.quickdlgs.*'

Engage Thrusters

1. Edit **ch7** | **Source Files** | **protected** | **views** | **job** | **index.php**.
   Correct the breadcrumb.

   ```
   $this->breadcrumbs=array(
     'Registered Jobs',
   );
   ```

   Change the action menu to link to the **Scheduled Job** list instead of the **Job List**,
   and remove the **Create Job** entry. We will be adding a button to perform this
   function shortly.

   ```
   $this->menu=array(
     array('label'=>'List Scheduled Jobs',
       'url'=>array('scheduledJob/index')),
   );
   ```

   Remove the ID field from the job grid.

271

The updated page will look like the following screenshot:
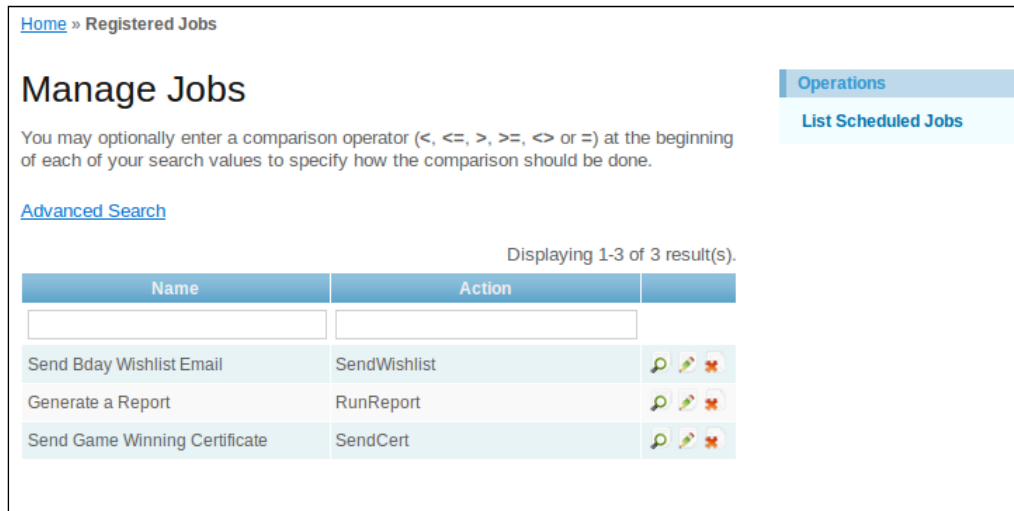
2. In the same file (**ch7** | **Source Files** | **protected** | **views** | **job** | **index.php**), add a `create` button, using the new extension `iframeButton` function, right before the grid is generated.

```php
<div class="right">
  <?php
    EQuickDlgs::iframeButton(
      array(
        'controllerRoute' => 'create',
        'dialogTitle' => 'Create item',
        'dialogWidth' => 800,
        'dialogHeight' => 275,
        'openButtonText' => 'Register New Job',
        'closeButtonText' => 'Close',
        'closeOnAction' => true, //important to invoke the
          close action in the actionCreate
        'refreshGridId' => 'job-grid', //the grid with this id
        will be refreshed after closing
      )
    );
  ?>
</div>
<?php $this->widget('zii.widgets.grid.CGridView', array(
```

The screen with the new **Register New Job** button will look like the following screenshot:

You can reload the page and try the button at this point. It will produce a modal dialog that contains the entire job creation page. Also, if you create a record, the grid will not update to indicate that a new record has been added.

3. To make the name field fit nicely into the modal dialog, change the size of the field in the form **ch7** | **Source Files** | **protected** | **views** | **job** | **_form.php**.

```
<?php echo $form->textField($model,'name',array
  ('size'=>40,'maxlength'=>64)); ?>
```

4. We will still access the **Create Job** screen from the **Job Scheduling** page, because it might be convenient to add a job in the midst of scheduling jobs. To reuse the same view for both the modal dialog and the full-page view, we will want to maintain the existing information and make it look nice in modal form.

5. Start by changing the action menu to contain links back to the **Registered Jobs** list and the **Scheduled Jobs** list in **ch7** | **Source Files** | **protected** | **views** | **job** | **create.php**.

```
$this->menu=array(
  array('label'=>'List Registered Jobs',
    'url'=>array('job/index')),
  array('label'=>'List Scheduled Jobs',
    'url'=>array('scheduledJob/index')),
);
```

Change the index label in the breadcrumbs to show that it is for **Registered Jobs**.

```
$this->breadcrumbs=array(
  'Registered Jobs'=>array('index'),
  'Create',
);
```

Change the header on the page from **H1 – Create Job** to **H6 – Register Job**.

```
<h6>Register Job</h6>
```

6. To stop the site template from rendering in the create dialog, replace the render call in the `actionCreate` function in the Job Controller (`ch7 | Source Files | protected | controllers | JobController.php`) to `EQuickDlgs::render`, which will detect the source of the call as an Ajax request and `renderPartial` the create view.

```
EQuickDlgs::render( 'create',array(
  'model'=>$model,
));
```

Also, add brackets around the if save condition, add a call to `checkDialogJsScript`, and change the redirect to the admin view.

```
if($model->save()) {
  EQuickDlgs::checkDialogJsScript();
  $this->redirect(array('admin'));
}
```

Now, when you click on **create** from the grid, the create dialog will look like this:

7. We will make the update view modal, also, so you can make the same changes to `actionUpdate` in the Job Controller. The resulting function will look like the following code snippet:

```
public function actionUpdate($id)
{
  $model=$this->loadModel($id);

  // Uncomment the following line if AJAX validation is
  //needed
  // $this->performAjaxValidation($model);

  if(isset($_POST['Job']))
  {
    $model->attributes=$_POST['Job'];
    if($model->save()) {
      EQuickDlgs::checkDialogJsScript();
      $this->redirect(array('admin','id'=>$model->id));
    }

  }

  EQuickDlgs::render('update',array(
    'model'=>$model,

  ));

}
```

8. The modal update will not appear until we change the buttons in the job index view. Replace the array that contains `CButtonColumn` with the following array that uses `EJuiDlgsColumn` in **ch7** | **Source Files** | **protected** | **views** | **job** | **index.php**.

```
'columns'=>array(
  'name',
  array(
    'class'=>'EJuiDlgsColumn',
    'updateDialog'=>array(
      'dialogWidth' => 580,
      'dialogHeight' => 250,
    ),
    'viewDialog'=>array(
      'dialogWidth' => 580,
      'dialogHeight' => 250,
    ),
  ),
),
```

9. We must change the **Update** view like we changed the **Create** view to make the modal appearance nicer and the page view (although we are not currently using it) clearer.

Change the action menu to display links back to the **Registered Jobs** list and the **Scheduled Jobs** list, as well as **Create** and **View** actions.

```
$this->menu=array(
  array('label'=>'Create Job', 'url'=>array('create')),
  array('label'=>'View Job', 'url'=>array
    ('view', 'id'=>$model->id)),
  array('label'=>'List Registered Jobs',
    'url'=>array('job/index')),
  array('label'=>'List Scheduled Jobs',
    'url'=>array('scheduledJob/index')),
);
```

Change the index label in the breadcrumbs to show that it is for **Registered Jobs**.

```
$this->breadcrumbs=array(
  'Registered Jobs'=>array('index'),
  $model->name=>array('view','id'=>$model->id),
  'Update',
);
```

Change the header on the page from H1 to H6.

```
<h6>Update Job <?php echo $model->id; ?></h6>
```

10. Finally, change the job controller view action to produce a modal result.

```
EQuickDlgs::render( 'view',array(
  'model'=>$this->loadModel($id),
));
}
```

Objective Complete - Mini Debriefing

We have fleshed out the job controls and consolidated them into a single page using Ajax, CGridView, and the `quickdlgs` extension.

Adding Job Scheduling

In this task, we will clean up and customize the scaffolded job scheduling pages, to make it easier for us to add and test job scheduling and processing in the following tasks.

Prepare for Lift Off

1. Download the Yii extension timepicker from `http://www.yiiframework.com/extension/timepicker/`.

   ```
   cd ~/Downloads

   wget http://www.yiiframework.com/
     extension/timepicker/files/timepicker5.zip
   ```

2. Unzip the package in your project's `extensions` directory.

   ```
   cd ~/projects/ch7/protected/extensions/

   unzip ~/Downloads/timepicker5.zip
   ```

3. Add the following entry to the import array in **ch7** | **Source Files** | **protected** | **config** | **main.php**.

   ```
   'ext.timepicker.*'
   ```

Engage Thrusters

1. Start by adding a class variable named `job_name` to the `ScheduledJob` model in **Source Files** | **protected** | **models** | **ScheduledJob.php**. This change is similar to the work we did on the user form in *Project 3*, *Access All Areas – Users and Logins*.

   ```
   class ScheduledJob extends CActiveRecord
   {
     public $job_name;
   ```

2. We will add a job drop-down to the scheduled job form, but before we do that, we have made enough drop-down functions to generalize our work. Create a class file in the `components` directory (**Source Files | protected | components**) named `SelectableActiveRecord.php` containing the following:

    ```php
    <?php

        class SelectableActiveRecord extends CActiveRecord {
          public function getOptions()
          {
            return CHtml::listData($this-
              >findAll(),'id','name');
          }

        }
    ```

3. Edit **Source Files | protected | models | Job.php**. Change the base class from `CActiveRecord` to `SelectableActiveRecord`.

    ```php
    class Job extends SelectableActiveRecord
    ```

4. We will do a quick retro-fit of Grade and Type models to use this base class. Remove the function `getGradeOptions` from **Source Files | protected | models | Grade.php** and the function `getTypeOptions` from **Source Files | protected | models | Type.php**. Change base class of each from `CActiveRecord` to `SelectableActiveRecord`. Then, update **Source Files | protected | views | book | _form.php** and change the drop-down list of both Grade and Type to use `getOptions`.

5. Now update the form for adding a scheduled job. Open **Source Files | protected | views | scheduledJob | _form.php**.

    Move the `job_id` field to the top of the page and change the field type for the `job_id` field from `textField` to `dropDownList` using the new `getOptions` function.

    ```php
    <?php echo $form->labelEx($model,'job_id'); ?>
    <?php echo $form->dropDownList($model, 'job_id',
      Job::model()->getOptions()); ?>
    <?php echo $form->error($model,'job_id'); ?>
    ```

    Change the field type for the active field from `textField` to `checkbox`.

    ```php
    <?php echo $form->labelEx($model,'active'); ?>
    <?php echo $form->checkbox($model,'active'); ?>
    <?php echo $form->error($model,'active'); ?>
    ```

Replace the `scheduled_time` field with the `timepicker` widget.

```
<div class="row">
  <?php echo $form->labelEx($model,'scheduled_time');
    ?>
  <?php $this->widget('ext.timepicker.timepicker', array(
    'model'=>$model,
    'name' => 'scheduled_time',
    'options'=> array(
      'dateFormat' =>'yy-mm-dd',
      'altFormat' =>'yy-mm-dd',
    ),
  ));
  ?>
  <?php echo $form->error($model,'scheduled_time'); ?>
</div>
```

Remove the fields `output`, `started`, and `completed` from the form. These fields will be updated by the job running mechanism.

The resulting form should look like the following screenshot:

6. Set the default value of `active` on a new scheduled job to `true` by adding the following line to the Create action in the Scheduled Job Controller in **Source Files | protected | controllers | ScheduledJobController**:

```
// set the default value of active to true
$model->active = true;
$this->render('create',array(
   'model'=>$model,
));
```

7. Remove the link for **Manage ScheduledJob** from **Source Files | protected | views | scheduledJob | create.php** and from **Source Files | protected | views | scheduledJob | update.php**.

8. Change the scheduled job view to show valid actions. In **Source Files | protected | views | scheduledJob | view. php** remove the link for **Manage ScheduledJob**.

Change the breadcrumb to show the job name.

```
$this->breadcrumbs=array(
   'Scheduled Jobs'=>array('index'),
   $model->job->name,
);
```

Also, change the header of the page to show the job name.

```
<h1>View Scheduled Job <?php echo $model->job->name;
   ?></h1>
```

Finally, replace `job_id` with the job name and move it to the top of the attribute list, and replace active with a string value of `true` or `false`.

```
'attributes'=>array(
  array (
    'name' => 'job_name',
    'header' => 'Job',
    'value' => $model->job->name,
  ),
  'params',
  'output',
  'scheduled_time',
  'started',
  'completed',
  array (
    'name' => 'active',
```

```
        'header' => 'Active',
        'value' => $model->active ? 'true' : 'false',
    ),
),
```

The finished screen should look like the following:

9. Finally, we will streamline the scheduled job index to show a few fields that we care about. Edit **Source Files | protected | views | scheduledJob | index.php** and remove the `id`, `params`, and `output` fields from the index view grid. Replace the `job_id` field with the job name.

```
array (
    'name' => 'job_name',
    'header' => 'Job',
    'value' => '$data->job->name',
),
```

Replace the `action` field with a checkbox column.

```
array (
    'class'=>'CCheckBoxColumn',
    'id' => 'active',
    'header' => 'Active',
    'checked' => '$data->active',
    'selectableRows' => 0,
),
```

The resulting screen will look as follows:

Objective Complete - Mini Debriefing

In this section, we customized the scheduled job screens to make it easier to input and manage jobs as we test our job scheduling system.

Adding Job Processing

In this task, we will create a simple job consumption script. We will use cron to run the script every so often. When it runs, it will query for jobs that need to run in this time frame, queue them up, execute them, and record the results. We want the jobs to run within the context of Yii, because we want to:

▸ Use the same database configuration as our web application

▸ Take advantage of Yii's libraries, primarily the database access utilities

To do this, our job processing script will be written as a Yii command, like the RBAC command we used in *Project 4*, *Level Up! Permission Levels*.

Engage Thrusters

1. Create a project directory named `utils`. Right-click on **ch7** | **Source Files** | **protected**, select **New** | **Folder**, and enter `utils`. We will keep the job entry script and any other administrative scripts we create, in this directory.

2. In the `utils` directory, create a custom Yii entry script named `job_entry.php`.

```php
<?php
  // change the following paths if necessary
  $yii='/opt/lampp/htdocs/yii/framework/yii.php';
  $config=dirname(__FILE__).'/../config/main.php';

  // remove the following lines when in production mode
  defined('YII_DEBUG') or define('YII_DEBUG',true);
  // specify how many levels of call stack should be
  //shown in each log message
  defined('YII_TRACE_LEVEL') or
    define('YII_TRACE_LEVEL',0);
  require_once($yii);
  $app = Yii::createConsoleApplication($config)->run();
?>
```

Our custom entry script allows us to execute console commands within the context of our Yii application. It differs from the project's `yiic.php`, for example, in that it uses the main application configuration file, and has a separate debugging and trace level.
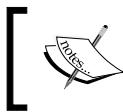
> We set the `YII_TRACE_LEVEL` variable in `job_entry.php` to `0`, so that the logging output we configure later will only report our log statements, and no additional trace lines of context.

3. In the `commands` directory, create a file named `JobProcessorCommand.php`. Right-click on **ch7** | **Source Files** | **protected** | **commands**, select **New** | **PHP File**, and enter `JobProcessorCommand`. This will be our job consumption script, and we will use the `job_entry` script to run it from the command line within our Yii context. Paste this skeleton command to initialize the job processor:

```php
<?php
  class JobProcessorCommand extends CConsoleCommand
  {
    private function getJobs() {
      return ScheduledJob::model()->findAll();
    }

    public function run($args)
    {
      $jobs = $this->getJobs();
      foreach ($jobs as $job) {
```

```
        echo "Running " . $job->job->name . " scheduled
          for " . $job->scheduled_time . "\n";
      }
    }
  }

?>
```
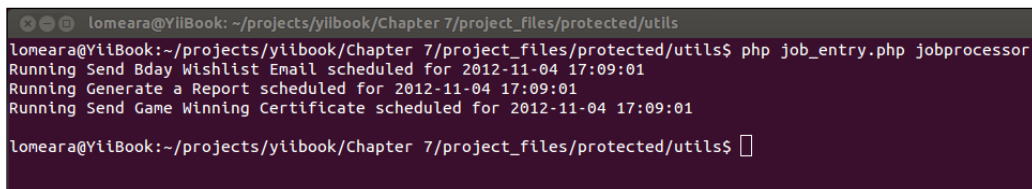
For now, to demonstrate how job processing will run in the context of Yii, the script prints a list of jobs. Try running it by running the following commands from a terminal window:

cd ~/projects/ch7/protected/utils/

php job_entry.php jobprocessor

The script will produce a list of scheduled jobs that look like the following:

4. Next, we want to narrow down the list of jobs to just active jobs scheduled for the current time. To help run this query, we will add named scopes to the `ScheduledJob` model. Named scopes provide a named query criteria that can be applied to an active record query. Edit **Source Files | protected | models | ScheduledJob.php** and add the following function:

```
public function scopes()
{
  return array(
    'active' => array(
      'condition' => 'active=1 AND completed IS NULL',
    ),
    'current' => array(
      'condition' => 'scheduled_time < now()',
    ),
  );
}
```

5. Apply the `active` scope in our job processor by changing the scheduled job query to use the active scope.

```
private function getJobs() {
  return ScheduledJob::model()->active()->findAll();
}
```

When you run the `jobprocessor` command, the output shows only the jobs marked active and not yet complete.

6. Add the `current` scope to the scheduled job query in the job processor.

```
private function getJobs() {
    return ScheduledJob::model()->active()->current()-
        >findAll();
}
```

Now, the `jobprocessor` command output shows only active jobs, not yet marked complete, scheduled for the current time or earlier.

```
lomeara@YiiBook:~/projects/yiibook/
  Chapter 7/project_files/protected/utils$ php
  job_entry.php jobprocessor
Running Send Bday Wishlist Email scheduled for
  2012-08-12 21:52:47
```

7. At this point, we will configure logging, so that we can capture information as our jobs are run. Add a log route array for this configuration in **ch7 | Source Files | protected | config | main.php**. The array will contain a value for categories, so that we only apply this route to log entries for the `jobprocessor` category. It will also contain a value for `fileName`, to write the job log output to its own file `job.log`, separate from the web application output `application.log`. We set the level to `Info`, `Error`, and `Warning` so that all of these levels are written to the file. Our logging statements will be `Info` level.

```
'log'=>array(
  'class'=>'CLogRouter',
  'routes'=>array(
    array(
      'class'=>'CFileLogRoute',
      //'levels'=>'trace, error, warning',
      'levels'=>'error, warning',
    ),
    array(
      'class'=>'CFileLogRoute',
      'levels'=>'info, error, warning',
      'logFile'=>'job.log',
      'categories'=>'jobprocessor',
    ),
```

285

8. Now change the run command in **ch7** | **Source Files** | **protected** | **commands** | **JobProcessorCommand.php** to run the action specified in the job. We will supply the action with the given parameters. We will also write a log entry recording the job run, and store the json-encoded output and time started and completed. Note, this will temporarily break the command until we make some more changes.

```php
public function run($args)
{
  $jobs = $this->getJobs();
  foreach ($jobs as $job) {
    Yii::log("Running - Job [" . $job->job->name .
      "] Action [" . $job->job->action .
      "] Parameters [" . $job->params .
      "] scheduled for " . $job->scheduled_time,
    'info', 'jobprocessor');
    $name = $job->job->action;
    $job->started = new CDbExpression('NOW()');
    $job->save();
    $job->output = json_encode($this->$name
      ($job->params));
    $job->completed = new CDbExpression('NOW()');
    $job->save();
  }
}
```

9. To complete the job processing system, set up system automation to run the job processor every so often.

   For example, on Unix, you could use crontab and schedule job processing once a night. Use the following command to open your user `crontab` for editing:

   crontab -e

   Select an editor, for example vim, and add the following line:

   ```
   15 2 * * * php ~/projects/protected/utils/job_entry.php
     jobprocessor
   ```

   Cron will run the job processor and subsequently all of the jobs scheduled for the previous day each night at 2:15 a.m.

   > You can learn more about crontab online and try out different configurations for running your job processor at different times or more frequently.

Objective Complete - Mini Debriefing

We have created a script to run our jobs within the context of our web app. It allows us to use our own project models and libraries in our job functions. We created a log configuration just for our job processor. Alternately, we could have created another config script for the job processor and passed that in `job_entry.php`. We demonstrated one way to configure our server to run our jobs every night.

Classified Intel

If you schedule many jobs, you may run into problems processing them; either because a job is very resource-intensive or many jobs are configured to run or some combination of these situations. To handle more job processing, you may want to look into adding multi-threaded support to your job processing. There are several ways to do this. One way to do this is to run them through the web server. We chose not to do this, because we did not want to permit our jobs to be run directly from the web. If you want to take this route, the jobs would either be web-accessible, or you could implement a token-based system to prevent unauthorized access to the jobs. See *Project 5*, *Service Please – Integrating Service Data*, for examples of using tokens and a discussion of secure token generation.

Creating and Registering a Job

The first job we will create will simply demonstrate how to create and call a job. It will not take parameters as input. It will not produce output. It will just run and send e-mails to the users of our system.

Prepare for Lift Off

To demonstrate this function, we have added an e-mail field to the user table. The e-mail field is roughly supported in the interface. You may want to expand on the support with validation and search features. For now, the create and update user screens will allow a user to view and edit the field.

The job that we are about to create will send an e-mail to all of your users. In order to send e-mails, you must have a mail server configured on your system.

If you do have a configured e-mail server, or you do not know whether or not you do, be careful about the e-mail addresses that are configured for your users. You may want to reduce the number of users in your database, and change any live e-mail addresses to fake e-mail addresses. `@email.com` is a good test e-mail domain to use.

If you do not have a configured e-mail server, you can still run the job function and verify the output in the job log file. We will create a reporting job later with more exciting output.

287

Engage Thrusters

1. You should have only one active, current job entry, `Send Bday Wishlist Email`, from the default schema data. If that is not the case, update the information in the scheduled job table to match, so that your jobs run correctly in the following steps. You can do this quickly by dropping the table, recreating it, and reloading the original job schema file **Source Files | protected | data | jobs.sql**.

2. The job we are creating will e-mail our wishlist to our friends. We want to use a list of users that does not include our account (admin), so let's add a scope to the user model.

    ```
    public function scopes()
    {
      return array(
        'not_admin' => array(
          'condition' => "username!='admin'",
        ),
      );
    }
    ```

3. Finally, create the job action to run. Add the following private function to **Source Files | protected | commands | JobProcessorCommand.php**:

    ```
    private function SendWishlist() {
      // prepare the email message
      $subject = "Some Gift Ideas";
      $headers = 'From: My CBDB Admin
        admin@mycbdb.com' . "\r\n" .
      'Reply-To: My CBDB Admin admin@mycbdb.com' . "\r\n" .
      'X-Mailer: PHP/' . phpversion();
      $email = "Here are a few of my gift wishes:\n";
      // build the body of the email
      $wishes = Wish::model()->findAll();
      foreach ($wishes as $w) {
        $email .="\t" . $w->title . "\n";
      }

      $email .="Please come to my website to see more
        about " .
      "my collection and play some games.";
      Yii::log("My wishlist email message is
        [" . $email. "]",
      'info', 'jobprocessor');

      $wishgivers = User::model()->not_admin()->findAll();
    ```

```
    foreach ($wishgivers as $wg) {
      Yii::log("Sending wishlist to " . $wg->username,
      'info', 'jobprocessor');
      //mail($email, $subject, $body, $headers);
    }
  }
```

We have commented out the actual command to send the e-mail in our job. When you run it, the only output will be log entries that are created. If you have a mail server set up and are comfortable sending an e-mail to all of your users, you can uncomment it and run it.

4. To run this job, you can schedule the `Send Bday Wishlist Email` job for some time soon and change the time your job processor will run. Alternatively, you can run the job manually with the following command:

**php ~/projects/ch7/protected/utils/job_entry.php
  jobprocessor**

If you want to run the job repeatedly, you will need to reset the completed timestamp in the scheduled job record to `NULL` after each run. We did this by executing a MySQL command to reset the value. Alternately, you could add a function to your interface to support repeated testing.

5. Check the log output in **Source Files** | **protected** | **runtime** | **job.log** to confirm that the job ran successfully. It should look something like the following:

```
2012/08/15 03:45:01 [info] [jobprocessor] Running -
  Job [Send Bday Wishlist Email] Action [SendWishlist]
  scheduled for 2012-08-12 21:52:47
2012/08/15 03:45:01 [info] [jobprocessor] My wishlist
  email message is [Here are a few of my gift wishes:
  Moebius' Airtight Garage Vol.1
  The Squiddy Avenger
  another great title
Please come to my website to see more about my
  collection and play some games.]
2012/08/15 03:45:01 [info] [jobprocessor]
  Sending wishlist to borrower
2012/08/15 03:45:01 [info] [jobprocessor]
  Sending wishlist to afriend
2012/08/15 03:45:01 [info] [jobprocessor]
  Sending wishlist to twg
2012/08/15 03:45:01 [info] [jobprocessor]
  Sending wishlist to tcreate
```

Objective Complete - Mini Debriefing

In this section we created a simple e-mail sending job, scheduled, and tested it.

Classified Intel

When you want to run a job many times to debug it, remember to reset the scheduled job entry in the database. If you do not, the job processor will see the scheduled job as already run and will not pick it up and run it. All you need to do is set `completed` to `null`, but you could set `completed`, `started`, and `output` to `null`. The following is a MySQL command you can use to set those values to null for all scheduled jobs:

```
UPDATE scheduled_job SET completed=null,started=null,
  output=null;
```

Creating a Graphical Report

Now we will write a job that generates JSON report data that can be displayed graphically. We chose to use the Flot JavaScript library (`http://code.google.com/p/flot/`) to present our report data, but you could write a report to output any reporting format you like. You could even generate an Excel spreadsheet.

Our report will produce a bar graph of our books by grade. We will set up Flot in the next task. For now, we will concentrate on writing a query and storing the JSON output.

Engage Thrusters

1. In the web app, edit the **Generate a Report** scheduled job, and change the scheduled time. You can use the **Now** button to quickly change the value to a time that will soon have past. Make sure that the `active` field is set to `1`.

2. Open **Source Files | protected | commands | JobProcessorCommand.php** and create a private function named `RunReport`.

   ```
   private function RunReport() {
   }
   ```

3. Add a query to get the number of books by grade.

   ```
   $criteria= new CDbCriteria();
   $criteria = array(
     'select' => 'count(grade_id) as num_grade, grade_id',
     'with' => array( 'grade' ),
     'group' => 'grade_id',
   );
   $books = Book::model()->findAll($criteria);
   ```

4. We will not be able to access the `num_grade` value until we add that field to the Book model (**Source Files** | **protected** | **models** | **Book.php**).

```
class Book extends CActiveRecord
{
  public $borrower_fullname = '';
  public $borrower_fname;
  public $borrower_lname;
  public $num_grade;
```

5. Add the following code after the query in the `RunReport` function to initialize the Flot report.

```
// initialize report
$report = array(
  'data'=> array (
    array(
      'label'=> 'Comic Books by Grade',
      'data'=>array(),
      'bars'=>array(
        'show'=>true,
        'align'=>'center',
      ),
    ),
  ),
  'options'=>array(
    'legend'=>array(
      'show'=>false,
    ),
  ),
  'htmlOptions'=>array(
    'style'=>'width:200px;height:200px;'
  )
);
```

6. After the report initialization, add the following `for` loop to iterate over the query results and add them to the report as data points. Return the result.

```
foreach ($books as $book) {
  $report['data'][0]['data'][] = array
    ($book->grade_id,$book->num_grade);
  $report['options']['xaxis']['ticks'][] = array
    ($book->grade_id,$book->grade->name);
}

return $report;
```

7. Run the report from the command line to capture the data.

291

Objective Complete - Mini Debriefing

For now, we have a reporting job that silently runs and collects the current grading status of our comic book collection. In the next section, we will display the results.

Displaying Graphical Report Output

In the previous section, we collected and prepared data. In this section, we simply have to add a graphing extension to display it. We chose to use the Flot extension because we liked the look and ease of use of the extension.

Prepare for Lift Off

1. Download the Yii extension EFlot from `http://www.yiiframework.com/extension/flot/`.

   ```
   cd ~/Downloads
   wget http://www.yiiframework.com/
     extension/flot/files/EFlot.zip
   ```

2. Unzip the package in your project's `extensions` directory.

   ```
   cd ~/projects/ch7/protected/extensions/
   unzip ~/Downloads/EFlot.zip
   ```

3. Add the following entry to the import array in **ch7** | **Source Files** | **protected** | **config** | **main.php**.

   ```
   'ext.EFlot.*'
   ```

Engage Thrusters

1. Update the scheduled job view, **Source Files** | **protected** | **views** | **scheduledJob** | **view.php**, to check for output data. If output is not null, pass the JSON-decoded data to the EFlot widget.

   ```php
   <?php
     if ($model->output != null) {
       $this->widget('application.extensions.
         EFlot.EFlotGraphWidget',
         json_decode($model->output, true)
       );
     }
   ?>
   ```

2. Open the scheduled job view for the **Generate a Report** job to see the results.
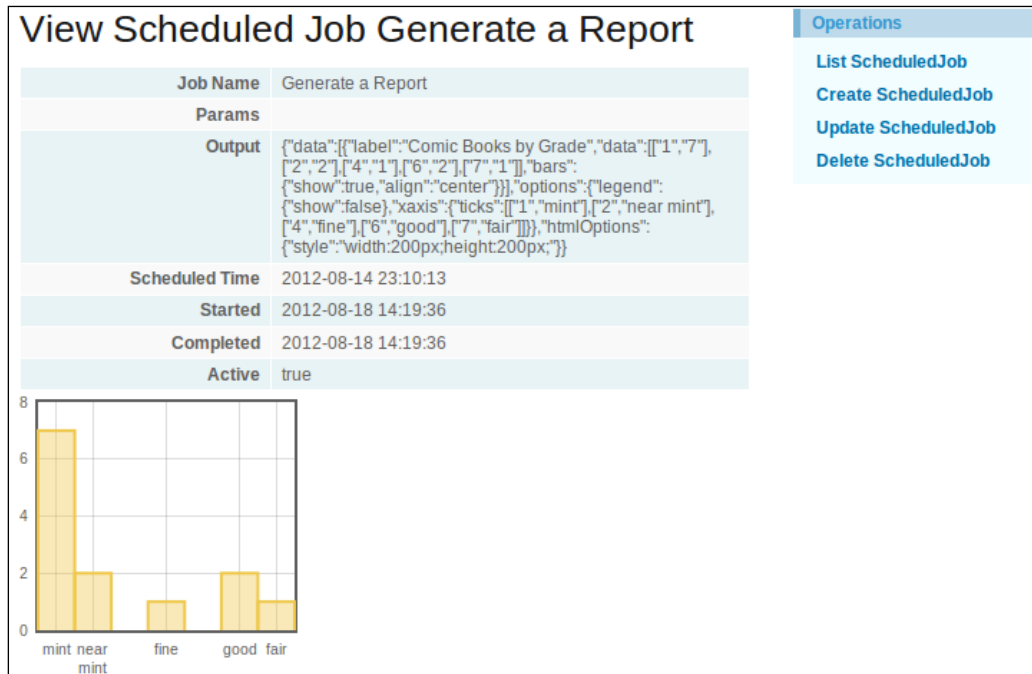
Objective Complete - Mini Debriefing

Our graph applies a minimal number of the features available. You may want to explore the options you can give Flot to produce different labels and charts. You also have the capability to add more jobs that query our data and prepare graphical reports.

Mission Accomplished

In this project, we have built a system to input, schedule, and process jobs. We have included support for running reporting jobs that generate graphical output and viewing their results.

You Ready to go Gung HO? A Hotshot Challenge

Here are some ideas to extend on the work from this chapter:

- ► Support priority in the job queue – both recording the priority for a scheduled job and applying the priority when choosing which jobs to run.

- ► Add support for scheduling recurring jobs.

- ► Replace job processing cron script with a full-time job processing daemon.

- ► Add Ajax to the job-scheduling grid to set jobs as active/inactive.

- ► Create more reporting jobs to try out the different types of reports you can create.

- ► Update the output format to include flags that indicate what type of data is stored and how it should be generated. Use this to support the EFlot format and some other formats, for example simple text output.

Project 8

Extend Yourself – Make a Module for Reuse

In this chapter, we will package the job queue function that we created in *Project 7*, *Let It Work While You Sleep – Reports and Job Queues*, into a module so we can share it and reuse it in future work. In the process, we will cover what makes a good module, how to put a module together, and how to post a module to the Yii community.

Mission Briefing

You will find yourself reusing some entities and functions in project after project. If you wanted to include that work in multiple projects without a framework for doing so, you would have to:

- ▶ Review your past projects to find the pieces you want
- ▶ Copy out each piece (model, view, controller, and supporting files)
- ▶ Integrate your old work into your current project
- ▶ Somehow replicate any changes or improvements to the shared work

Yii provides a facility for gathering your work into reusable packages, such as modules, widgets, and components, that you can plug into projects as needed.

In this project, we will cover:

- How to identify good candidates for reuse
- How to package the files into a module
- How to test your module to make sure you can use it successfully
- How to submit your module to the Yii community to share your work with the world
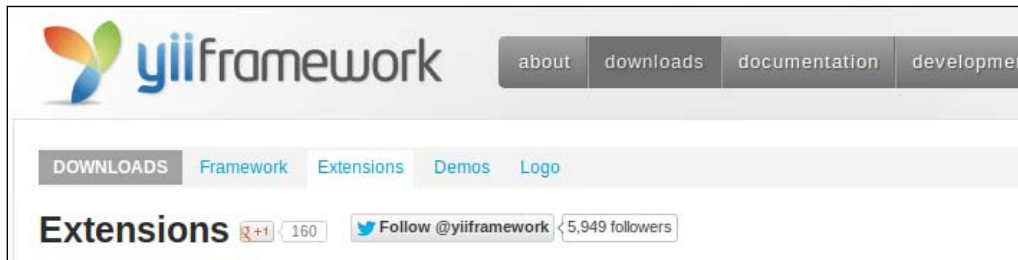
Why Is It Awesome?

What isn't awesome about writing a module?

- You save yourself from having to write the same tasks over and over again
- You can maintain cross-project code in one place and deploy updates easily
- You can share your work with the rest of the world

It will take a little work to create a module from some of the functions in your project, but the time you will save reusing that module over and over again will more than make up for the effort.

Your Hotshot Objectives

- Selecting Code for Reuse
- Preparing Your Module Framework
- Moving Your Module Files
- Writing a Migration Script
- Re-incorporating Your Module
- Testing Your Module
- Submitting Your Module

Mission Checklist

This project assumes that you have a web development environment prepared. If you do not have one, the tasks in *Project 1*, *Develop a Comic Book Database*, will guide you through setting one up. In order to work this project, you will need to set up the project files that have been provided with the book. Refer to the *Preface* of the book for instructions on downloading these files. The files for this project include a Yii project directory with a database schema. To prepare for the project, follow these steps, replacing the username `lomeara` with your own username.

1. Copy the project files into your working directory.

   ```
   cp -r ~/Downloads/project_files/Chapter\ 8/project_files
     ~/projects/ch8
   ```

2. Make the directories that Yii uses web writeable.

   ```
   cd ~/projects/ch8/
   ```

   ```
   sudo chown -R lomeara:www-data protected/runtime assets protected/
   models protected/controllers protected/views
   ```

3. Create a link in the `webroot` directory to the copied directory.

   ```
   cd /opt/lampp/htdocs
   ```

   ```
   sudo ln -s ~/projects/ch8 cbdb
   ```

4. Import the project into NetBeans (remember to set the project URL to `http://localhost/cbdb`) and configure for Yii development with PHPUnit.

5. Create a database named `cbdb` and load the database schema (`~/projects/ch8/protected/data/schema.sql`) into it.

6. If you are not using the XAMPP stack or if your access to MySQL is password protected, you should review and update the Yii configuration file (in NetBeans: **ch8** | **Source Files** | **protected** | **config** | **main.php**).

> The admin login to the website is admin/test.

Selecting Code for Reuse

Not everything we write is a good candidate for reuse. We are going to talk about some ways you can identify and isolate code for reuse. The *Engage Thrusters* section in this task is more of a checklist to review than a list of steps to take, but we will apply the checklist to our job module as we go.

Engage Thrusters

1. Does the function operate on isolated tables in the database?

   For example, although many tables in your database may refer to the user table, the user information can be isolated to a user table containing username and password with some extension tables for things such as personal and contact information. It is a good candidate, because you typically know from the start if a project will need user management. In which case, you can include a user management module at the beginning of a project and build from there.

   If this is not the case, and your module does require some table information, you are not necessarily halted in your tracks. You could include support for module configuration so that users can identify required table information. For an example of a module that does this, see the RBAC module that we used in *Project 4*, *Level Up! Permission Levels*.

   Our proposed module uses the `tables` job and `scheduled_job` from *Project 7*, *Let It Work While You Sleep – Reports and Job Queues*. The schema for `job` and `scheduled_job` look like the following:
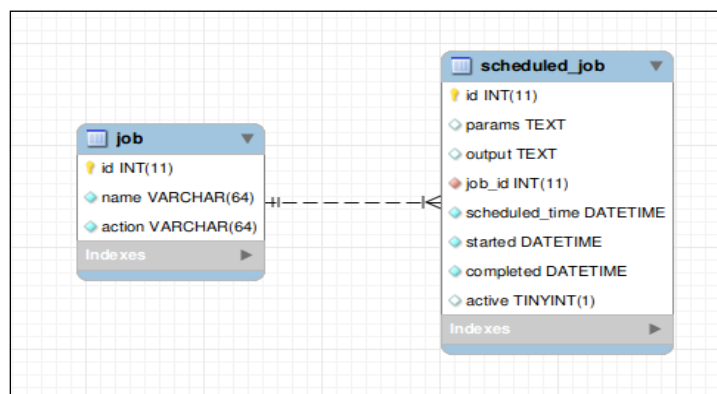
   The `scheduled_job` table depends on the `job` table, but no other tables are required. We are in good shape here.

2. Can you easily identify the models, controllers, and views that will be a part of your module?

   Similar to the table isolation, if your core objects are isolated, they are more easily incorporated into a module. And, again, if they are not, you may be able to address the dependencies by providing a way to configure the module.

The models we use for jobs are `Job.php` and `ScheduledJob.php`. As we know there are no table dependencies, so we can be pretty sure there are no model dependencies. And, when we review the files, we find that there are none.

The controllers are `JobController.php` and `ScheduledJobController.php`. They do not depend on other controllers or classes. They both inherit from the default Yii Controller class. This class may not be included in a project that would use the module. You can address this by:

- ❑ Including the Controller class in your module, so that your controllers can inherit from it
- ❑ Reworking your controllers to not inherit from a base class

We are going to take the first option, and include the base controller class in our module.

It takes more time to go through the view files, because there are so many. You want to make sure that only standard Yii functions are used.

The views for our proposed job module, `job` and `scheduledJob`, do have dependencies. They use three extensions: `quickdlgs`, `flot`, and `timepicker`. If your module depends on someone else's work, check the license for their work. `quickdlgs` and `timepicker` are under the BSD 2 license. `flot` has a different, but similar license. We can redistribute the extensions in our module if we adhere to the terms of their licenses.

Alternately, we could note these dependencies in the documentation for our module, but for the sake of making an easily-testable, stand alone module, we will include all of the required extensions.

3. Are there any complementary classes or utilities that should be included in your module?

   Maybe you created some additional classes or scripts that your functions use. Be sure to identify these and include them.

   Our job queue relies on a utility script, **ch8** | **Source Files** | **protected** | **utils** | **job_entry.php**, and a command script, **ch8** | **Source Files** | **protected** | **commands** | **JobProcessorCommand.php**.

4. Does the code represent a task that is common across your projects?

   User management, contact management, comments, ratings. All of these modules are already implemented for Yii and provide common web application functionality. If you implement a common generic function, such as these, it is probably going to be worth your while to make a module out of it. But, of course, if you are thinking about implementing a common generic function, such as user management, contact management, and so on, check first to see if a module that meets your needs is already available.

299

We searched Yii extensions (`http://www.yiiframework.com/extensions/`) for extensions related to scheduling jobs. We found some extensions for creating cron jobs. Our job manager does things a little differently, so we decided to move forward with implementing it and packaging it for reuse.

Objective Complete - Mini Debriefing

In this section, we reviewed the conditions and qualifications for creating a module. We looked at the tables, models, views, and controllers that make up our job management utility. We found no dependencies that require special handling, so we are ready to begin making a module.
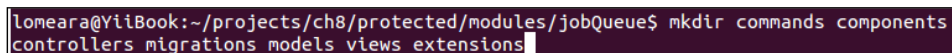
Preparing Your Module Framework

The first task in creating a module is creating a space for the module. In this task, we will set up the framework that our module will inhabit.

Engage Thrusters

1. In the modules directory of your project (**ch8** | **Source Files** | **protected** | **modules**) create a directory named `jobQueue`.

2. In the newly created `jobQueue` directory, create the following directories:

    ❑ `commands`

    ❑ `components`

    ❑ `controllers`

    ❑ `migrations`

    ❑ `models`

    ❑ `views`

    ❑ `extensions`

    The Unix command to create these directories looks like the one shown in the following screenshot:

3. A module must include in its root directory a class that extends from `CWebModule`. In the main module directory (**ch8** | **Source Files** | **protected** | **modules** | **jobQueue**), create a file named `JobQueueModule.php` and input the following contents:

```php
<?php

class JobQueueModule extends CWebModule
{
  public function init()
  {
    $this->setImport(array(
      'jobQueue.models.*',
      'jobQueue.extensions.quickdlgs.*',
      'jobQueue.extensions.timepicker.*',
      'jobQueue.extensions.EFlot.*',
      'jobQueue.components.*',
    ));
  }
}
```

This is the file where you would access and apply the configuration values that we mentioned in the *Selecting Code for Reuse* task.

Classified Intel

If you wanted to include configurable variables in your module, here is how you would do it:

1. In the base module class, which extends `CWebModule`, add class variables for any configuration fields you want to include in your module. Be sure to provide a default value for your variable. For example, if you wanted to have a configuration variable named `jobQueueUser`, we would add a variable to the class as follows:

```php
class JobQueueModule extends CWebModule
{
  /* @var $jobQueueUser String The name of the job
  queue user*/
  public $jobQueueUser = "lomeara";
```

2. Now, when you want to change the configuration value for the module in your project, edit the main project configuration file. For example to override the default value for the previous jobQueueUser variable, we would add the following entry to **ch8** | **Source Files** | **protected** | **config** | **main.php**:

```php
'modules'=>array(
  'jobQueue' => array(
    'jobQueueUser'=>'www-data',
```

Objective Complete-Mini Debriefing

The result of this task is a directory framework to hold our module. We can add module, view, and controller files. We can also expand on the module class, if we need to add configuration or custom behaviors.

Moving Your Module Files

An unavoidable task in our module creation is corralling our files into our module directory. We decided to multi-task and remove the files from our project while we place them in the module directory. In other words, we are just going to move the files from our project into the module. This is going to break the functionality temporarily. In a later task, we will take the necessary steps to make the job queue work again.

Engage Thrusters

1. Move the model files, `Job.php` and `ScheduledJob.php`, from your project model directory **ch8** | **Source Files** | **protected** | **models** into your module model directory **ch8** | **Source Files** | **protected** | **modules** | **jobQueue** | **models**.

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/models$
   mv ../../../models/Job.php .
   ```

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/models$
   mv ../../../models/ScheduledJob.php .
   ```

2. Move the controller files, `JobController.php` and `ScheduledJobController.php`, from your project controller directory **ch8** | **Source Files** | **protected** | **controllers** into your module controller directory **ch8** | **Source Files** | **protected** | **modules** | **jobQueue** | **controllers**.

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/
   controllers$ mv ../../../controllers/JobController.php .
   ```

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/
   controllers$ mv ../../../controllers/ScheduledJobController.php .
   ```

3. Move the view directories, `job` and `scheduledJob`, from your project view directory **ch8** | **Source Files** | **protected** | **views** into your module view directory **ch8** | **Source Files** | **protected** | **modules** | **jobQueue** | **views**.

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/views$
   mv ../../../views/job/ .
   ```

   ```
   lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/views$
   mv ../../../views/scheduledJob/ .
   ```

4. Move all of the directories from your project extension directory **ch8 | Source Files | protected | extensions** into your module extension directory (**ch8 | Source Files | protected | modules | jobQueue | extensions**).

```
lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/
extensions$ mv ../../../extensions/* .
```

```
lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/
extensions$ ls
```

```
EFlot   quickdlgs   timepicker
```

5. Copy the Controller class from the component directory **ch8 | Source Files | protected | components** into your module component directory **ch8 | Source Files | protected | modules | jobQueue | components**.

```
lomeara@YiiBook:~/projects/ch8/protected/modules/jobQueue/c
  omponents$ cp ../../../components/Controller.php .
```

6. Remove the configuration entries for the extensions from the project configuration file **ch8 | Source Files | protected | config | main.php**, The import array, with the extension entries removed, should look like the following:

```
'import'=>array(
  'application.models.*',
  'application.components.*',
  'application.modules.srbac.
    controllers.SBaseController',
  'application.modules.auditTrail.models.AuditTrail',
),
```

7. Move the utility directory from your project directory **ch8 | Source Files | protected | utils** into your module directory **ch8 | Source Files | protected | modules | jobQueue**.

8. Move the `JobProcessorCommand` script from the `commands` directory in your project **ch8 | Source Files | protected | commands | JobProcessorCommand.php** to the `commands` directory in your module **ch8 | Source Files | protected | modules | jobQueue | commands | JobProcessorCommand.php**.

9. Create a console configuration directory **ch8 | Source Files | protected | modules | jobQueue.** Create a file in the module config directory **ch8 | Source Files | protected | modules | jobQueue | config | console.php** with the following contents:

```
<?php
return array(
  'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
  'runtimePath'=>dirname(__FILE__)
    .DIRECTORY_SEPARATOR.'../../../runtime',
```

303

```
        'preload'=>array('log'),
        'components'=>array(
          'log'=>array(
            'class'=>'CLogRouter',
            'routes'=>array(
              array(
                'class'=>'CFileLogRoute',
                'levels'=>'info, error, warning',
                'logFile'=>'job.log',
                'categories'=>'jobprocessor',
              ),
            ),
          ),
        ),
      );
```

10. The resulting module directory tree should look this:
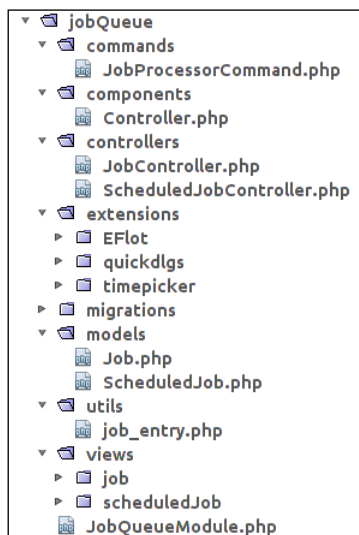
Objective Complete - Mini Debriefing

In this task, we moved all of the files related to our job module from our project directory
into the `jobQueue` directory.

Writing a Migration Script

The migration script will make any database changes that our module requires. Since our module depends on the existence of two tables, we will write a migration script to create them.

Engage Thrusters

1. To create a new migration, change to the project directory. Run Yiic with the `migrate` command, specifying the path alias to the module migrations directory.

    `cd ~/projects/ch8/protected`

    `php yiic.php migrate create create_tables_job_queue`
    `--migrationPath=application.modules.jobQueue.migrations`

    The command will output a file named something like this:

    `m120927_012345_create_tables_job_queue.php`

    The filename is the letter `m`, followed by the UTC timestamp of the time the file is created, followed by the name you gave the command.

    The contents of the file will look something like the following code snippet:

    ```php
    <?php

    class m120927_012345_create_tables_job_queue extends CDbMigration
    {
      public function up()
      {
      }

      public function down()
      {
        echo "m120927_012345_create_tables_job_queue
          does not support migration down.\n";
        return false;
      }

      /*
      // Use safeUp/safeDown to do migration with transaction
      public function safeUp()
      {
      }

      public function safeDown()
      {
      }
      */
    ```

305

2. The migration file that we generated is just a skeleton. Now we need to flesh it out with the actual migration steps. The up method holds the migration steps. The down method contains the steps to revert the migration, if they can be reverted. The generated code in the down function is for the case where a migration cannot be reverted.

   Let's start by completing the `up` function. We will need to create both the `job` and `scheduled_job` tables. And, actually, we will implement the `safeUp` function, because our database supports transactions. Replace `up` with the following code snippet:

```
public function safeUp()
{
  $this->createTable('job', array(
    'id' => 'pk',
    'name' => 'varchar(64) NOT NULL',
    'action' => 'varchar(64) NOT NULL',
  ));
  $this->createTable('scheduled_job', array(
    'id' => 'pk',
    'params' => 'text',
    'output' => 'text',
    'job_id' => 'int(11) NOT NULL',
    'scheduled_time' => 'datetime NOT NULL',
    'started' => 'datetime NOT NULL',
    'completed' => 'datetime NOT NULL',
    'active' => 'tinyint(1) DEFAULT \'0\'',
  ));
  $this->createIndex( 'job_id', 'scheduled_job',
    'job_id');
  $this->addForeignKey('scheduled_job_ibfk_1',
    'scheduled_job', 'job_id', 'job', 'id');
}
```

   This function will create the related tables we have already been using: `job` and `scheduled_job`, as well as the relationship from `scheduled_job` to `job`. Because we have used the transaction safe method, the contents of the function will be wrapped in a transaction. If your function included any database commands that do not carry an implicit commit, such as `insert`, `update` or `delete`, those steps would be rolled back if any step fails.

3. Before we try the script, let's get ready for testing. Since our migrate up function only creates tables, we can have a migration down function that will undo the change. All it has to do is drop the tables. It will look like the following code snippet:

```
public function safeDown()
{
  $this->dropTable( 'scheduled_job' );
  $this->dropTable( 'job' );
}
```

4. In order to test the migration script we have just written, we must drop the tables that we have been using from the database. If you have some entries in those tables that you might want to use later, we recommend that you back up your database first, and then enter the following commands in the NetBeans MySQL command window:

```
drop table scheduled_job;
```

```
drop table job;
```

5. Test the migrate up function, which will re-create the tables, by running the following commands in a terminal window:

```
cd ~/projects/ch8/protected
```

```
php yiic.php migrate up --migrationPath=application.modules.
jobQueue.migrations
```

6. If your `migrate up` command was successful, try the `migrate down` function in the same terminal window.

```
php yiic.php migrate down --
  migrationPath=application.modules.jobQueue.migrations
```

If you receive an error message, such as **… create_tables_job_queue does not support migration down**, check your `safeDown` function. Make sure that it does not end with a return false.

Don't forget to migrate back up before you continue.

Objective Complete - Mini Debriefing

We have just created a migration script, which will be a part of our module, and prepare a database for use by our module code.

Classified Intel

Migration scripts are useful for more than just modules. They are also useful for maintaining the database in your project. You can use them in upgrades, deployments, and team development. Start your project by creating the initial migration script, as we just did. Each time you want to make a change to your database schema, create a new migration script to record and propagate the changes. Using this method to update your schema will result in consistent deployments and will help disseminate changes when you are developing in a team.

Re-incorporating Your Module

Now that we have our module directory initialized and populated, and we have created the necessary migration script, let's reincorporate what we have back into our project, to verify that it still works well.

Engage Thrusters

1. In your web browser, log back into your web app `http://localhost/cbdb/`.

2. Log in and navigate to the job screen **Admin | Jobs** and see that we get an error. For the moment, we have broken our job queue.
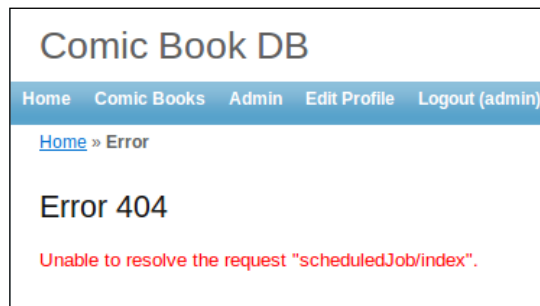
3. Yii no longer knows where to find the job queue functions. One problem is that the link is now in the module's name space, so we need to update the menu to link to the new place. Edit **ch8 | Source Files | protected | views | layouts | main.php**. Change the entry for jobs to include the `jobQueue` directory in its path.

```
array('label'=>'Users', 'url'=>array('/user/index')),
array('label'=>'Jobs',
    'url'=>array('/jobQueue/scheduledJob/index'),
    'authItemName' => 'Authority'),
array('label'=>'Reports', 'url'=>array('/report/index')),
```

4. Update Yii's information, by editing the project configuration file **ch8** | **Source Files** | **protected** | **config** | **main.php**. Add an entry to the modules array:

```
'jobQueue'=>array(),
```

5. Reload the **Jobs** page and you'll see a new error **Access Denied**.

Oh! That's because we are using RBAC and our permissions no longer match the path to the controller. To fix this, navigate to the srbac administrative interface, and add `jobQueue@` to the beginning of all job-related operations. Here is a list so you don't forget any:

- `jobQueue@JobAdmin`
- `jobQueue@JobCreate`
- `jobQueue@JobDelete`
- `jobQueue@JobIndex`
- `jobQueue@JobList`
- `jobQueue@JobUpdate`
- `jobQueue@JobView`
- `jobQueue@ScheduledJobAdmin`
- `jobQueue@ScheduledJobCreate`
- `jobQueue@ScheduledJobDelete`
- `jobQueue@ScheduledJobIndex`
- `jobQueue@ScheduledJobUpdate`
- `jobQueue@ScheduledJobView`

By updating the values in the existing operations, instead of creating new ones, we don't have to recreate our existing authorization rules.

6. Now, if you reload your page, you can navigate to **Admin | Jobs**, and you can click to the scheduled job index successfully.

Oh, but the action menu doesn't display!

Well, you can get to it if you enter the URL, but you will see an error page that looks like the following screenshot:
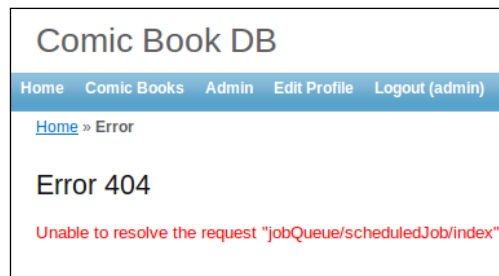
7. Add an entry to each menu item for `'authItemName' => 'Authority'` as follows:

```
$this->menu=array(
  array('label'=>'Schedule Job', 'url'=>array('create'),
    'authItemName' => 'Authority'),
  array('label'=>'List Registered Jobs',
    'url'=>array('job/index'), 'authItemName' =>
    'Authority'),
  array('label'=>'Register Job',
    'url'=>array('job/create'), 'authItemName' =>
    'Authority'),
);
```

310

You will need to make this change in every job view file that has an action menu. Here is another list for your reference:

❏ `views/scheduledJob/create.php`

❏ `views/scheduledJob/index.php`

❏ `views/scheduledJob/update.php`

❏ `views/scheduledJob/view.php`

❏ `views/job/create.php`

❏ `views/ job /index.php`

❏ `views/ job /update.php`

❏ `views/ job /view.php`

Now the menus appear like they should.

8. Well… except for the Job Index, which has a nasty error.
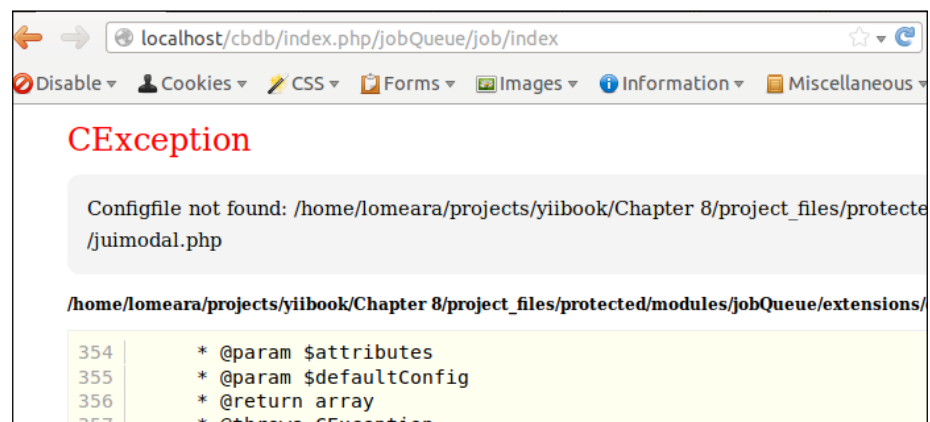
We need to fix the routes in our buttons on the Job Index.

9. To fix the create button, change the `controllerRoute` attribute in the `iFrameButton` function call in **ch8 | Source Files | protected | modules | jobQueue | views | job | index.php**.

    ```
    EQuickDlgs::iframeButton(
      array(
    'controllerRoute' => 'jobQueue/job/create',
        'dialogTitle' => 'Create item',
    ```

10. Also we have to make a minor change to the `quickdlgs` extension, so that it will work within a module. Edit **ch8 | Source Files | protected | modules | jobQueue | extensions | quickdlgs | EQuickDlgs.php** and change the `EXTINSTALLDIR` constant to be the module extension directory alias.

    ```
    const EXTINSTALLDIR =
      'jobQueue.extensions.quickdlgs';
    ```

    Now the Job Index page will display correctly.
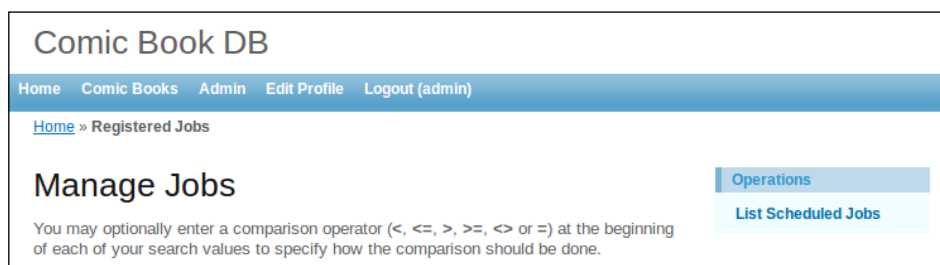
11. If you try to schedule a job, you will encounter another error.

12. We need to correct the path alias for the `timepicker` widget. Change the reference in **ch8 | Source Files | protected | modules | jobQueue | views | scheduledJob | _form.php** to the following:

    ```
    <?php $this->widget('application.modules.
      jobQueue.extensions.timepicker.timepicker', array(
    ```

The fixed page will look like the following screenshot:
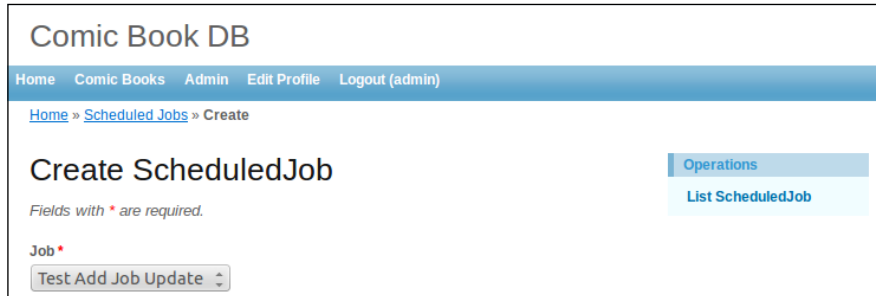
13. Finally, if you add an entry to the registered job list and try to view or edit from the registered job grid, those functions will not work.
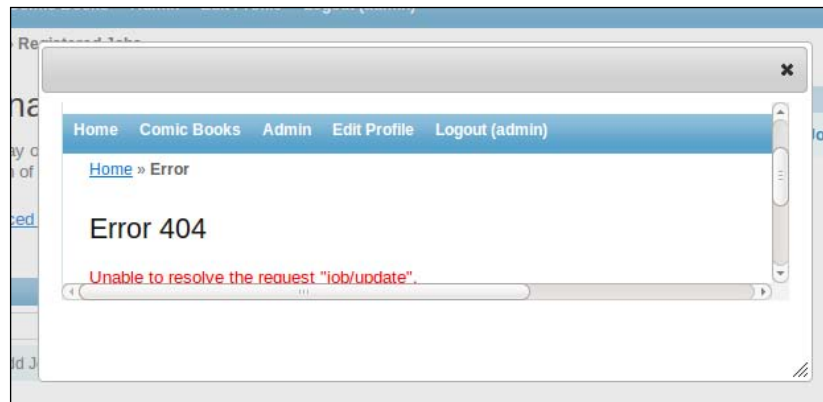
14. Update the configuration for the update and view dialogs in **ch8** | **Source Files** | **protected** | **modules** | **jobQueue** | **views** | **job** | **index.php** as follows:

```
 'updateDialog'=>array(
'controllerRoute' => 'jobQueue/job/update',
'actionParams' => array('id'=>'$data->id'),
   'dialogWidth' => 580,
   'dialogHeight' => 250,
 ),
 'viewDialog'=>array(
'controllerRoute' => 'jobQueue/job/view',
'actionParams' => array('id'=>'$data->id'),
   'dialogWidth' => 580,
   'dialogHeight' => 250,
 ),
```

313

Now the edit and update buttons will work as expected.

Since the delete dialog does not depend on an extension, it does not require any change.

Objective Complete - Mini Debriefing

We have just tried out the Job Queue functions in our web application to see how well they work with the new module. We made adjustments to the module file as we encountered configuration errors. The result is a functional module that fits into our site.

Testing Your Module

In the last task, we walked through the Job Queue screens to make sure that they work correctly. In this task, we will test the function of the module further with data.

Engage Thrusters

1. When we removed the tables from the database to test the migration script, we lost the job data that we had been using. Let's begin our testing by inputting the jobs that we used in *Project 7*, *Let It Work While You Sleep - Reports and Job Queues*.

    Delete any data you may have entered to test the forms.

2. Go to the Job Index (`http://localhost/cbdb/index.php/jobQueue/job/index`) and create the following entries:

    ❑   **Name**: **Send Bday Wishlist Email**

    Action: **SendWishlist**

    ❑   **Name**: **Generate a Report**

    Action: **RunReport**

The list of registered jobs should look like the following screenshot:

3. Now, go to the scheduled Job Index `http://localhost/cbdb/index.php/jobQueue/scheduledJob/index` and schedule these jobs (click on the link for **Schedule Job** in the **Operations** menu) to run in the past (so that when you run the job processor, the job will be sure to run). The queue should look like the following screenshot:

4. Try running the job processor, like we did in *Project 7, Let It Work While You Sleep - Reports and Job Queues*, except we will be running the script from the module directory. Open a terminal window and try running the following command:

```
php ~/projects/ch8/protected/modules/jobQueue/
  utils/job_entry.php jobprocessor
```

Looks like we have another error to correct; we must correct the path to the configuration file. For our module, we want to use the new console configuration file that we created.

```
$config=dirname(__FILE__).'/../config/console.php';
```

Objective Complete - Mini Debriefing

We have worked out the last few kinks in our module by inputting some example jobs and running them to verify that all the pieces work correctly together.

Submitting Your Module

We have now walked through all of the functions our module performs. Everything looks great. Before packing it up and sharing it, we should implement some automated unit tests. Or better yet, we should have started with the unit tests before we implemented any of the functions. Assuming that a full unit test suite has been created, applied to the module, and passed with flying colors, we are now ready to submit the module to the Yii website to share with the world.

Engage Thrusters

1. Create an account in the Yii forum `http://www.yiiframework.com/`.

2. Post your extension to the Yii forum to gather feedback from other users. If many people become interested in your module, this step will really put your module through the ringer.

3. Determine what license you will use for your module.

4. Prepare the documentation for your module.

    ❑   Prepare screenshots of your module in action

    ❑   Include any configuration instructions

    ❑   Gather a list of all required software

    ❑   Compress your module directory into a tarball or ZIP file or both

    ❑   Write installation instructions

    ❑   Consolidate usage examples that will help your users understand how to incorporate your module

5. Go to the Yii Extensions page `http://www.yiiframework.com/extension/` and click on the **Create extension** button.

6. Upload your compressed module package.

7. Input your module description, license information, usage information, installation instructions, and so on.

8. Submit your module for review and sharing.

Objective Complete - Mini Debriefing

After you have followed these steps, you will be the proud, community contributor of a Yii module that anyone in the world can download and use, depending on the terms of your license. Good work! What are you going to accomplish next?

Mission Accomplished

In this project, we learned how to identify pieces of our work that could benefit our projects or others by being converted into reusable modules. We demonstrated the module creation process by converting a function that we had written in a previous chapter, complete with required extensions and custom scripts, into a module structure. We tested our new module in place against our web application project and then prepared and submitted the module to the wider community.
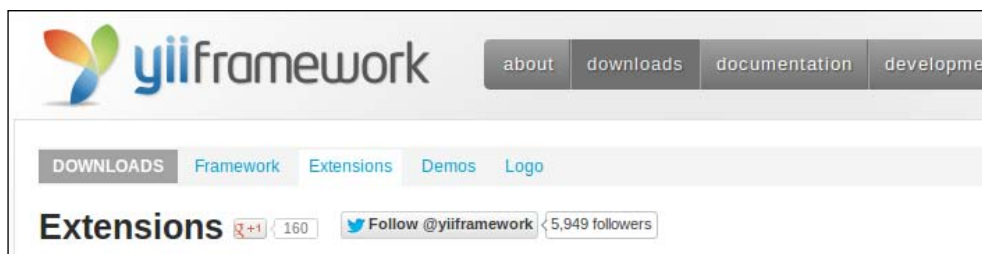
You Ready to go Gung HO? A Hotshot Challenge

Create your own module for reuse! Here are some ideas of modules you could write to benefit your own projects and the Yii community:

- Address manager
- Contact information manager
- Customer information manager
- Leaderboard for game sites
- Shopping cart
- Wish list
- Any API you commonly use

    - Any Google API (Google Maps, Google+, or Fusion Tables)
    - Amazon web services
    - Twitter REST API
    - Any credit card processor
    - Any shipping vendor

Index

Symbols

Thank you for buying
Yii Rapid Application Development Hotshot

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.

open source
community experience distilled
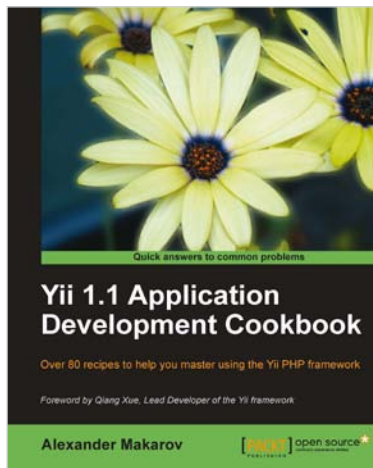
Yii 1.1 Application Development Cookbook

ISBN: 978-1-84951-548-1   Paperback: 392 pages

Over 80 recipes to help you master using the Yii
PHP framework

1. Learn to use Yii more efficiently through plentiful
   Yii recipes on diverse topics

2. Make the most efficient use of your controller and
   views and reuse them

3. Automate error tracking and understand the Yii log
   and stack trace
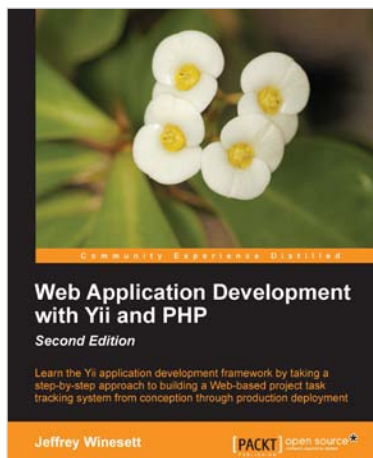
Web Application Development with Yii and PHP

ISBN: 978-1-84951-872-7   Paperback: 332 pages

Learn the Yii application development framework by
taking a step-by-step approach to building a Web-based
project task tracking system from conception through
production deployment

1. A step-by-step guide to creating a modern Web
   application using PHP, MySQL, and Yii

2. Build a real-world, user-based, database-driven
   project task management application using the
   Yii development framework

3. Start with a general idea, and finish with deploying
   to production, learning everything about Yii
   inbetween, from "A"ctive record to "Z"ii
   component library

Please check **www.PacktPub.com** for information on our titles

Responsive Web Design with HTML5 and CSS3

ISBN: 978-1-84969-318-9   Paperback: 324 pages

Learn responsive design using HTML5 and CSS3 to adapt websites to any browser or screen size

1. Everything needed to code websites in HTML5 and CSS3 that are responsive to every device or screen size

2. Learn the main new features of HTML5 and use CSS3's stunning new capabilities including animations, transitions and transformations

3. Real world examples show how to progressively enhance a responsive design while providing fall backs for older browsers
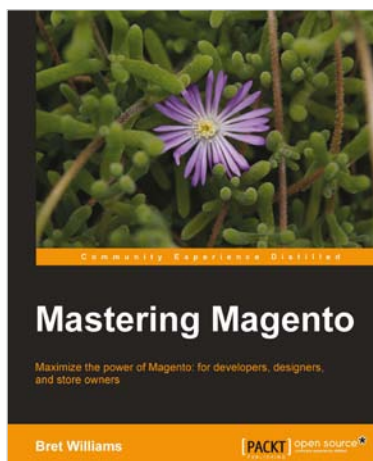
Mastering Magento

ISBN: 978-1-84951-694-5   Paperback: 300 pages

Maximize the power of Magento: for developers, designers, and store owners

1. Learn how to customize your Magento store for maximum performance

2. Exploit little known techniques for extending and tuning your Magento installation.

3. Step-by-step guides for making your store run faster, better and more productively

Please check **www.PacktPub.com** for information on our titles