Web Application Development with Yii and PHP

Chapter No. 3
"The TrackStar Application"
In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 "The TrackStar Application"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Jeffrey Winesett has over ten years of experience building large-scale, web-based applications. He has been a strong proponent of using open source development frameworks when developing applications, and a champion of the Yii framework in particular since its initial alpha release. He frequently presents on, writes about, and develops with Yii as often as possible.

I would like to thank Qiang Xue for creating this amazing framework, and the entire Yii framework development team who continually improve and maintain it. I thank all of the technical reviewers, editors, and staff at Packt Publishing for their fantastic contributions, suggestions, and improvements.

I would also like to thank my family, who have provided encouragement and unconditional support, and to my many colleagues over the years for providing invaluable opportunities for me to explore new technologies, expand my knowledge, and shape my career.

For More Information:
Web Application Development with Yii and PHP

This book is a step-by-step tutorial to developing a real-world application using the Yii web application development framework. The book attempts to mirror the environment of a software development team that is being tasked with building an online application, touching on each aspect of the software development lifecycle, as you build a project task management application from its conception to its production deployment.

After a brief, general introduction to the Yii framework, and going through the iconic "Hello World" example, the remaining chapters break down in the same way as software development iterations do in real-world projects. We begin with creating a working application with a valid, tested connection to a database.

We then move on to defining our main database entities and domain object model, and become familiar with Yii's Object Relational Mapping (ORM) layer Active Record. We learn how to lean on Yii's code generation tools to automatically build our create/read/update/delete (CRUD) functionality against our newly created models. We also focus on how Yii's form validation and submission model works.

By the end of Chapter 5, Managing Issues, you will have a working application that allows you to manage the projects and issues (tasks) within those projects.

We then turn to the topic of user management. We learn about the built-in authentication model within Yii to assist with the application login and logout functionality. We take a deep dive into the authorization model, first taking advantage of Yii's simple access control model and then implementing the more sophisticated, role-based access control (RBAC) framework that Yii provides.

By the end of Chapter 7, User Access Control, all of the basics of a task-management application are in place. The next several chapters begin to focus on additional user features, user experience, and design. We add user comment functionality, introducing a reusable, content portlet architecture approach in the process. We add in an RSS web feed and demonstrate how easy it is to integrate other third-party tools and frameworks within a Yii application. We take advantage of Yii's theming structure to help streamline and design the application, and then introduce Yii's internationalization (I18N) features so that the application can be adapted to various languages and regions without engineering changes.

In the final chapter, we turn our focus to preparing the application for production deployment. We introduce ways to optimize performance and improve security, to prepare the application for a production environment.

For More Information:
What This Book Covers

Chapter 1, Meet Yii, provides you with a brief history of Yii, an introduction to the Model View Controller (MVC) application architecture, and you are introduced to the typical request life cycle as it makes its way from the end user through the application, and finally as a response back to the end user.

Chapter 2, Getting Started, is dedicated to downloading and installing the framework, creating a new Yii application shell, and introducing Gii, Yii's powerful and flexible code generation tool.

Chapter 3, The TrackStar Application, is where the TrackStar application is introduced. This is an online, project-management and issue-tracking application that you will be building throughout the remaining chapters. Here you learn how to connect a Yii application to an underlying database. You also learn how to run the interactive shell from the command line. The last part of this chapter is focused on providing an overview of unit and functional testing within a Yii application, and provides a concrete example of writing a unit test in Yii.

Chapter 4, Project CRUD, helps you to start interacting with the database, to begin adding features to the database-driven Yii application TrackStar. You learn how to use Yii Migrations for database change management, and we use the Gii tool to create model classes and also to build out create, read, update, and delete (CRUD) functionality using the model class. The reader is also introduced to configuring and performing form field validation in this chapter.

Chapter 5, Managing Issues, explains how additional related database tables are added to the TrackStar application, and how the reader is introduced to the relational Active Record in Yii. This chapter also covers using controller filters to tap into the application life cycle in order to provide pre-action and post-action processing. The official Yii extension library Zii is introduced, and we use Zii widgets to enhance the TrackStar application.

Chapter 6, User Management and Authentication, explains how to authenticate users in a Yii. While adding the ability to manage users in the TrackStar application, the reader learns to take advantage of Behaviors in Yii, which provide an extremely flexible approach to sharing common code and functionality across your Yii components. This chapter also covers the Yii authentication model in detail.

For More Information:
Chapter 7, User Access Control, is dedicated to Yii's authorization model. First we cover the simple access control features, which allow you to easily configure access rules for the controller actions that are based on several parameters. Then we look at how role-based access control (RBAC) is implemented in Yii, which allows a much more robust authorization model for complete access control based on hierarchical model of roles, operations, and tasks. Implementing role-based access control into the TrackStar application also introduced the reader to using the console command in Yii.

Chapter 8, Adding User Comments, helps demonstrate how to implement the feature for allowing users to leave comments on projects and issues in the TrackStar application; we introduce how to configure and use a statistical query relation, how to create highly reusable user interface components called Widgets, and how to define and use named scopes in Yii.

Chapter 9, Adding an RSS Web Feed, demonstrates how easy it is to use other third-party frameworks and libraries in Yii applications, and shows you how to use the URL management features of Yii to customize your applications' URL format and structure.

Chapter 10, Making It Look Good, helps you learn more about the views in Yii, and how to use layouts to manage markup and content shared across your application pages. Theming is also introduced, as we show how easy it is to give a Yii application a completely new look without having to alter any of the underlying engineering. We then take a look at internationalization (i18n) and localization (l10n) in Yii as language translation is added to our TrackStar application.

Chapter 11, Using Yii Modules, explains how to add administrative features to the TrackStar site by using a Yii module. Modules provide a very flexible approach to developing and managing the larger, self-contained sections of your application.

Chapter 12, Production Readiness, helps us prepare our TrackStar application for production. You learn about Yii's logging framework, caching techniques, and error-handling methods to help get your Yii applications production-ready.

For More Information:  
The TrackStar Application

We could continue to keep adding to our simple "Hello, World!" application to provide examples of Yii's features, but that won't really help to understand the framework in the context of a real-world application. In order to do that, we need to build towards something that will more closely resemble the types of applications web developers actually have to build. That is exactly what we are going to be doing throughout the rest of this book.

In this chapter, we will introduce the project task-tracking application called TrackStar. There are many other project management and issue-tracking applications out there in the world, and the basic functionality of ours will not be any different from many of these. So why build it, you ask? It turns out that this type of user-based application has many features that are common to many web applications. This will allow us to achieve two primary goals:

• Showcase Yii's incredible utility and feature set as we build useful functionality and conquer real-world web application challenges

• Provide real-world examples and approaches that will immediately be applicable to your next web application project

Introducing TrackStar

TrackStar is a software development life cycle (SDLC) issue-management application. Its main goal is to help keep track of the many issues that arise throughout the course of building software applications. It is a user-based application that allows the creation of user accounts and grants access to the application features once a user has been authenticated and authorized. It allows a user to add and manage projects.
Projects can have users associated with them (typically the team members working on the project) as well as issues. The project issues will be things such as development tasks and application bugs. The issues can be assigned to members of the project and will have a status such as not yet started, started, and finished. In this way, the tracking tool can give an accurate depiction of the projects with regard to what has been accomplished, what is currently in progress, and what is yet to be started.

Creating user stories

Simple user stories are a great way to identify the necessary feature functionality of an application. User stories, in their simplest form, state what a user can do with a piece of software. They should start simple, and grow in complexity as you dive into more and more of the details around each feature. Our goal here is to start with just enough complexity to allow us to get started. If necessary, we’ll add more detail later.

We briefly touched on the three main entities that play a large role in this application, namely users, projects, and issues. These are our primary domain objects and are extremely important items in this application. So let’s start with them.

Users

TrackStar is a user-based web application. At a high level, the user can be in one of two user states:

- Anonymous
- Authenticated

An anonymous user is any user of the application that has not been authenticated through the login process. Anonymous users will only have access to register for a new account or login. All other functionality will be restricted to authenticated users.

An authenticated user is any user that has provided valid authentication credentials through the login process. In other words, authenticated users are users that have logged in. Authenticated users will have access to the main feature functionality of the application, such as creating and managing projects, and project issues.

Projects

Managing the project is the primary purpose of the TrackStar application. A project represents a general, high-level goal to be achieved by one or more users of the application. The project is typically broken down into more granular tasks, or issues, that represent the smaller steps that need to be taken to achieve the overall goal.
As an example, let's take what we are going to be doing throughout this book, that is, building a project and issue-tracking management application. Unfortunately, we can't use our yet-to-be-created application as a tool to help us track its own development, but if we could, we might create a project named "Build The TrackStar Project/Issue Management Tool". This project would be broken down into more granular project issues, for example, "Create the login screen", "Design database schema for issues", and so on.

Authenticated users can create new projects. The creator of the project within an account will have a special role within that project called the **project owner**. Project owners have the ability to edit and delete these projects as well as add new members to the project. Other users associated with the project, besides the project owner, are referred to simply as **project members**. Project members will have the ability to add new issues as well as edit existing ones.

### Issues

Project issues will be classified into one of three categories:

- **Features**: Items that represent real features to be added to the application. For example, implementation of the login functionality.
- **Tasks**: Items that represent work that needs to be done, but are not an actual feature of the software. For example, setting up the build and integration server.
- **Bugs**: Items that represent application behaviors that are not working as expected. For example, the account registration form does not validate the format of an input e-mail address.

And issues can be in one of the following three status states:

- Not yet started
- Started
- Finished

Project members can add new issues to a project as well as edit and delete them. They can assign issues to themselves or other project members.

For now, this is enough information on these three main entities to allow us to move forward. We could go into a lot more detail about "what exactly does account registration entail?" or "how exactly does one add a new task to a project?", but we have outlined enough specifications to begin on these basic features. We'll nail down the more granular details as we proceed with the implementation.
However before we start, we should jot down some basic navigation and application workflow. This will help everyone to better understand the general layout and flow of the application we are building.

**Navigation and page flow**

It is always good to outline the main pages within an application and see how they fit together. This will help us quickly identify some needed Yii controllers, actions, and views, as well as help to set everyone’s expectations on what we’ll be building towards at the onset of our development.

The following diagram shows the basic application flow, from logging in through to the project details listing:

![Diagram of application flow](image)

When a user first comes to the application, they must log in and authenticate themselves prior to proceeding to any functionality. Once successfully logged in, they will be presented with a list of their current projects along with the ability to create a new project. Choosing a specific project will take them to the project details page. The project details page will present a list of the issues by type. There will also be the ability to add a new issue as well as edit any of the listed issues.
This is all pretty basic functionality, but the figure gives us a little more information on how the application is stitched together and allows us to start identifying our needed models, views, and controllers better. It also allows something visual to be shared with others so that everyone involved has the same picture of what we are working towards. In my experience, almost everyone prefers pictures over written specifications when first thinking through a new application.

**Data relationships**

We still need to think a little more about the data we will be working with as we begin to build toward these specifications. If we pick out all the main nouns from our system, we may end up with a pretty good list of domain objects and, by extension of using active record, the data we want to model. Our previously outlined user stories identify the following:

- A User
- A Project
- An Issue

Based on this and the other details provided in the user stories and application workflow diagram, a first attempt at the necessary data model is shown in the following diagram:
The TrackStar Application

This is a very basic object model that outlines our primary data entities, their respective attributes, and some of the relationships between them. The 1..* and 0..* on either side of the line between the project and user objects represents that there is a many-to-many relationship between them. A user can be associated with zero or more projects, and a project has one or more users. Similarly, we have represented the fact that a project can have 0 or more issues associated with it, whereas an issue belongs to just one specific project. Also, a user can be the owner of (or requestor of) many issues, but an issue just has one owner (also just one requestor).

We have kept the attributes as simple as possible at this stage. A user is going to need a username and a password in order to get past the login screen. The project has only a name attribute.

An issue has the most associated information based on what we currently know about them. As discussed briefly in the previously defined user stories, issues will have a type attribute to distinguish the general category (bug, feature, or task). They will also have a status attribute to indicate the progress of the issue being worked on. There will be a logged in user who initially creates the issue; this is the requestor. And once a user in the system has been assigned to work on the issue, they will be the owner of the issue. We have also defined the description attribute to allow for some descriptive text of the issue to be entered.

Note that we have not been explicitly talking about schemas or databases yet. The fact is that until we think through what is really needed from a data perspective, we won't know the right tool to use to house this data. Would flat files on the filesystem work just as well as a relational database? Do we need to persist data at all?

The answers to these questions are not always necessary at this early planning stage. It is better to focus more on the features we want and the type of data needed to support these features. We can turn to the explicit technology implementation details after we have had a chance to discuss these ideas with other project stakeholders, to ensure we are on the right track. Other project stakeholders include anyone and everyone involved in this development project. This can include the client, if you are building an application for someone else, as well as other development team members, product/project managers, and so on. It is always a good idea to get some feedback from "the team" to help validate the approach and any assumptions being made.

In our case, there is really no one else involved in this development effort. So, we can quickly draw some conclusions to answer our data-related questions and move forward with our application development.
Since this is a web-based application, and given the nature of the information we need to store, retrieve, and manipulate, we can conclude that it would be best to persist the data in this application. Also, based on the relationships that exist between the types of data we want to capture and manage, a good approach to storing this data would be in a relational database. Based on its ease of use, excellent price point, its general popularity among PHP application developers, and its compatibility with the Yii framework, we will be using MySQL as the specific database server.

Now that we know enough about what we are going to start building and how we are going to start building it, let's get started.

Creating the new application

First things first, let's get the initial Yii web application created. We have already seen how easy this is to accomplish in Chapter 2, Getting Started. As we did there, we will assume the following:

- **YiiRoot** is the directory where you have installed Yii
- **WebRoot** is configured as the document root of your web server (that is, to where http://localhost/ resolves)

So from the command line, change to your WebRoot directory and execute the following:

```
% YiiRoot/framework/yiic webapp trackstar
Create a Web application under '/WebRoot/trackstar'? [Yes|No] Yes
```

This provides us with our skeleton directory structure and our out-of-the-box working application. You should be able to view the home page of this new application by navigating to http://localhost/trackstar/index.php?r=site/index.

Because our default controller is SiteController and the default action within that controller is actionIndex(), we could also navigate to the same page without specifying the route.
Connecting to the database

Now that we have our skeleton application up and running, let's work on getting it properly connected to a database. In fact, the skeleton application was autoconfigured to use a database. A by-product of using the yiic tool is that our new application is configured to use a SQLite database. If you take a peek in the main application configuration file, located at protected/config/main.php, you will see the following declaration about halfway down:

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

And you can also verify the existence of protected/data/testdrive.db, which is the SQLite database it is configured to use.

Since we have already made the decision to use MySQL, we'll need to make some configuration changes. However, before we change the configuration to use a MySQL database server, let's briefly talk about Yii and databases more generally.

Yii and databases

Yii provides great support for database programming. Yii's Data Access Objects (DAO) are built on top of the PHP Data Objects (PDO) extension (http://php.net/pdo). This is a database abstraction layer that enables the application to interact with the database through a database-independent interface. All the supported database management systems (DBMS) are encapsulated behind a single uniform interface. In this way, the code can remain database independent and the applications developed using Yii DAO can easily be switched to use a different DBMS without the need for modification.

To establish a connection with a supported DBMS, you can simply create a new CDbConnection instance:

```php
$connection=new CDbConnection($dsn,$username,$password);
```

Here the format of the $dsn variable depends on the specific PDO database driver being used. Some common formats include:

- **SQLite**: sqlite:/path/to/dbfile
- **MySQL**: mysql:host=localhost;dbname=testdb
- **PostgreSQL**: pgsql:host=localhost;port=5432;dbname=testdb
- **SQL Server**: mssql:host=localhost;dbname=testdb
- **Oracle**: oci:dbname=//localhost:1521/testdb

For More Information:

CDbConnection also extends from CApplicationComponent, which allows it to be configured as an application component. This means that we can add it to the components property of the application, and customize the class and property values in the main configuration file. This is our preferred approach, which we will detail next.

**Adding a db connection as an application component**

Let's take a quick step back. When we created the initial application, we specified the application type to be a web application. Remember we specified webapp on the command line. Doing so specified that the application singleton class that is created upon each request be of the type CWebApplication. This Yii application singleton is the execution context within which all request processing is run. Its main task is to resolve the user request and route it to an appropriate controller for further processing. This was represented as the Yii Application Router back in the diagrams used in Chapter 1, Meet Yii, when we covered the request routing. It also serves as the central place for keeping application-level configuration values.

To customize our application configuration, we normally provide a configuration file to initialize its property values when the application instance is being created. The main application configuration file is located in /protected/config/main.php. This is a PHP file containing an array of key-value pairs. Each key represents the name of a property of the application instance, and each value is the corresponding property's initial value. If you open up this file, you will see that several settings have already been configured for us.

Adding an application component to the configuration is easy. Open up the file (/protected/config/main.php) and locate the components property.

We can see that there are entries already specifying a log and user application component. These will be covered in the subsequent chapters. We can also see (and as we noted previously) that there is a db component there as well, configured to use a SQLite connection to a SQLite database located at protected/data/testdrive.db. There is also a commented out section that defines this db component to use a MySQL database. All we need to do is remove the SQLite db component definition, uncomment the section that defines the MySQL component, and then make the appropriate changes to match your database name, username, and password so the connection can be made. The following code shows this change:

```php
// application components
'components'=>array(
  ...
  //comment out or remove the reference to the sqlite db
  /*
  'db'=>array(
```

The TrackStar Application

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

// uncomment the following to use a MySQL database
    'db'=>array(
        'connectionString' => 'mysql:host=localhost;dbname=trackstar',
        'emulatePrepare' => true,
        'username' => '[your-db-username]',
        'password' => '[your-db-password]',
        'charset' => 'utf8',
    ),

This assumes that a MySQL database has been created called trackstar and is available to connect using the localhost. Depending on your environment, you may need to specify the localhost IP of 127.0.0.1 rather than localhost. One of the great benefits of making this an application component is that now, anywhere throughout our application, we can reference the database connection simply as a property of the main Yii application $Yii::app()->db. Similarly, we can use this as a reference for any of the other components defined in the config file.

The charset property when set to 'utf8' sets the character set used for the database connection. This property is only used for MySQL and PostgreSQL databases. It will default to null, which means that it will use the default charset. We are setting it here to ensure proper utf8 unicode character support for our PHP application.

The emulatePrepare => true configuration sets a PDO attribute (PDO::ATTR_EMULATE_PREPARES) to true, which is recommended if you are using PHP 5.1.3 or higher. This was added to PHP 5.1.3, and when used, causes the PDO native query parser to be used rather than the native prepared statements APIs in the MySQL client. The native prepared statements in the MySQL client cannot take advantage of the query cache, and as such have been known to result in poor performance. The PDO native query parser can use the query cache, and so it is recommended to use this option if available (PHP 5.1.3 or higher).

So, we have specified a MySQL database called trackstar as well as the username and password needed to connect to this database. We did not show you how to create such a database in MySQL. We assume you understand how to set up a MySQL database and how to use it. Please refer to your specific database documentation if you are unsure of how to create a new database, called trackstar, and configure a username and password for connectivity.

For More Information:
Testing the database connection

Before we move on, we should ensure that our database connection is actually working. There are a few ways we could do this. We’ll look at two approaches. In the first approach, we’ll use the `yiic` command-line tool to start an interactive shell for our application and ensure that there are no errors when we attempt to reference the application db component. Then we’ll provide a second approach that will introduce us to unit testing in Yii with PHPUnit.

Using the interactive shell

We’ll start off with a simple test using the Yii interactive shell. As you recall, we used the `webapp` command along with the `yiic` command-line utility to create our new application. Another command to use with this utility is `shell`. This allows you to run PHP commands within the context of the Yii application, directly from the command line.

To start the shell, navigate to the root directory of the application, that is the directory that contains the `index.php` entry script `Webroot/trackstar/`. Then run the `yiic` utility, passing in `shell` as the command (refer to the following screenshot).

```
$/.protected/yiic shell
YII Interactive Tool v1.1 (based on YII v1.1.12)
Please type 'help' for help. Type 'exit' to quit.
>>
```

This starts the shell and allows you to enter the commands directly after the `>>` prompt.

What we want to do to test our connection is ensure that our database connection application component is accessible. We can simply `echo` out the connection string and verify that it returns what we set it to in our configuration. So from the shell prompt type the following:

```
>> echo Yii::app()->db->connectionString;
```

It should echo something similar to the following:

```
mysql:host=localhost;dbname=trackstar
```

This demonstrates that the db application component is configured correctly and available for use in our application.

For More Information:

Automated testing – unit and functional tests

Gathering feedback is of fundamental importance to application development; feedback from the users of the application and other project stakeholders, feedback from the development team members, and feedback directly from the software itself. Developing software in a manner that will allow it to tell you when something is broken can turn the fear associated with integrating and deploying applications into boredom. The method by which you can empower your software with this feedback mechanism is writing automated unit and functional tests, and then executing them repeatedly and often.

Unit and functional testing

Unit tests are written to provide the developer with verification that the code is doing the right things. Functional tests are written to provide the developer, as well as other project stakeholders, with verification that the application as a whole is doing things the right way.

Unit tests

Unit tests are the tests that focus on the smallest units within a software application. In an object-oriented application, such as a Yii web application, the smallest units are the public methods that make up the interfaces to the classes. Unit tests should focus on one single class and not require other classes or objects to run it. Their purpose is to validate that a single unit of code is working as expected.

Functional tests

Functional tests focus on testing the end-to-end feature functionality of the application. These tests exist at a higher level than the unit tests and typically require multiple classes or objects to run. Their purpose is to validate that a given feature of the application is working as expected.

Benefits of testing

There are many benefits to writing unit and functional tests. For one, they are a great way to provide documentation. Unit tests can quickly tell the exact story of why a block of code exists. Similarly, the functional tests document what features are implemented within an application. If you stay diligent in writing these tests, then the documentation continues to evolve naturally as the application evolves.

For More Information:

They are also invaluable as a feedback mechanism to constantly reassure the
developer and other project stakeholders that the code and application is working
as expected. You run your tests every time you make changes to the code and get
immediate feedback on whether or not something that you altered inadvertently
changed the expected behavior of the system. You can then address these issues
immediately. This really increases the confidence that developers have with the
application and translates to less bugs and more successful projects.

This immediate feedback also helps to facilitate change and improve the design of
the code. A developer is more likely to make improvements to existing code, if a
suite of tests are in place to immediately provide feedback as to whether the changes
made altered the application behavior. The confidence provided by a suite of unit
and functional tests allows developers to write better software, release more stable
applications, and ship quality products.

Testing in Yii
As of version 1.1, Yii is tightly integrated with the PHPUnit (http://www.phpunit.
de/) and Selenium Remote Control (http://seleniumhq.org/projects/remote-
control/) testing frameworks. You may certainly test Yii PHP code with any of
the testing frameworks available. However, the tight integration of Yii with the two
previously mentioned frameworks makes things even easier. And making things
easy is one of our primary goals here.

When we used the yiic webapp console command to create our new web
application, we noticed that many files and directories were automatically created
for us. The ones among these that are relevant to writing and executing automated
tests are the following:

<table>
<thead>
<tr>
<th>File/directory</th>
<th>Contains/Stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>trackstar/</td>
<td>Contains all the files listed in the file/directory column</td>
</tr>
<tr>
<td>protected/</td>
<td>Protected application files</td>
</tr>
<tr>
<td>tests/</td>
<td>Tests for the application</td>
</tr>
<tr>
<td>fixtures/</td>
<td>Database fixtures</td>
</tr>
<tr>
<td>functional/</td>
<td>Functional tests</td>
</tr>
<tr>
<td>unit/</td>
<td>Unit tests</td>
</tr>
<tr>
<td>report/</td>
<td>Coverage reports</td>
</tr>
<tr>
<td>bootstrap.php</td>
<td>The script executed at the very beginning of the tests</td>
</tr>
<tr>
<td>phpunit.xml</td>
<td>The PHPUnit configuration file</td>
</tr>
<tr>
<td>WebTestCase.php</td>
<td>The base class for web-based functional tests</td>
</tr>
</tbody>
</table>

For More Information:
www.packtpub.com/agile-web-application-development-with-yii-second-
edition/book
The TrackStar Application

You can place your test files into three main directories, namely fixtures, functional, and unit. The report directory is used to store the generated code coverage reports.

The PHP extension, XDebug, must be installed in order to generate reports. For details on this installation, see http://xdebug.org/docs/install. This extension is not required for the following example.

Unit tests

A unit test in Yii is written as a PHP class that extends from the framework class CTestCase. The conventions prescribe it to be named AbcTest, where Abc is replaced by the name of the class being tested. For example, if we were to test the MessageController class in our "Hello, World!" application from Chapter 2, Getting Started, we would name the test class MessageControllerTest. This class is saved in a file called MessageControllerTest.php under the directory protected/tests/unit/.

The test class primarily has a set of test methods named testXyz, where Xyz is often the same as the method name for which you are writing the test.

Continuing with the MessageController example, if we were testing our actionHelloworld() method, we would name the corresponding test method testActionHelloworld() in our MessageControllerTest class.

Installing PHPUnit

As of version 1.1, Yii is tightly integrated with the PHPUnit (http://www.phpunit.de/) testing framework.

In order to follow through with this example, you will need to install PHPUnit. This should be done using the Pear Installer. (For more information on Pear, see http://pear.php.net.) Please visit the following URL for more information on how to install PHPUnit based on your environment configuration:

https://github.com/sebastianbergmann/phpunit/

It is certainly beyond the scope of this book to specifically cover PHPUnit's testing features. It is recommended that you take some time to go through the documentation to get a feel for the jargon and for writing basic unit tests: https://github.com/sebastianbergmann/phpunit/

For More Information:
Testing the connection

Assuming you have successfully installed PHPUnit, we can add a test for our database connection under `protected/tests/unit/`. Let's create a simple database connectivity test file under this directory called `DbTest.php`. Add this new file with the following contents:

```php
<?php

class DbTest extends CTestCase
{
    public function testConnection()
    {
        $this->assertTrue(true);
    }
}
```

Here we have added a fairly trivial test. The `assertTrue()` method, which is a part of PHPUnit, is an assertion that will pass if the argument passed to it is `true`, and it will fail if it is `false`. In this case, it is testing if `true` is `true`. So this test will certainly pass. We are doing this to make sure that our new application is working as expected, for testing with PHPUnit. Navigate to the tests folder and execute this new test:

```
%cd /WebRoot/trackstar/protected/tests
%phpunit unit/DbTest.php
```

```
... Time: 0 seconds, Memory: 10.00Mb
OK (1 test, 1 assertion)
```

If for some reason this test failed on your system, you may need to change `protected/tests/bootstrap.php` so that the variable `$yiit` properly points to your `/YiiRoot/yiit.php` file.

Confident that our testing framework is working as expected within our newly created TrackStar application, we can use this to write a test for the db connection.

Change the `assertEquals(true)` statement in the `testConnection()` test method to:

```php
$this->assertNotNull(Yii::app()->db->connectionString);
```

For More Information:

And rerun the test:

```
%phpunit unit/DbTest.php
```

```
...
Time: 0 seconds, Memory: 10.00Mb

OK (1 test, 1 assertion)
```

As you recall, since we configured our database connection as an application component named `db`, `Yii::app()->db` should return an instance of the `CDbConnection` class. If the application failed to establish a database connection, this test would return an error. Since the test still passes, we can move forward with the confidence that the database connection is set up properly.

## Summary

This chapter introduced the task-tracking application, TrackStar, which we will be developing throughout the rest of this book. We talked about what the application is and what it does, and provided some high-level requirements for the application in the form of informal user stories. We then identified some of the main domain objects we will need to create, as well as worked through some of the data we will need to be able to house and manage.

We then took our first step towards building our TrackStar application. We created a new application with all of the working functionality that comes "for free" from the autogenerated code. We also configured our application to be connected to a MySQL database and demonstrated two approaches to test that connection. One approach demonstrated Yii’s integration with PHPUnit and how to write automated tests for your Yii application.

In the next chapter, we will get to finally sink our teeth into more sophisticated features. We will begin to do some actual coding as we implement the needed functionality to manage our project entities within the application.

For More Information:

Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.