



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Web Application Development with Yii 2 and PHP

Fast-track your web application development using the new
generation Yii PHP framework

Mark Safronov Jeffrey Winesett

[PACKT] open source*
PUBLISHING community experience distilled

Web Application Development with Yii 2 and PHP

Fast-track your web application development using
the new generation Yii PHP framework

Mark Safronov

Jeffrey Winesett



BIRMINGHAM - MUMBAI

Web Application Development with Yii 2 and PHP

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either expressed or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: August 2010

Second edition: November 2012

Third edition: September 2014

Production reference: 1190914

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

ISBN 978-1-78398-188-5

www.packtpub.com

Cover Image by Aniket Sawant (aniket_sawant_photography@hotmail.com)

Credits

Authors

Mark Safronov
Jeffrey Winesett

Project Coordinators

Venitha Cutinho
Akash Poojary

Reviewers

Ajay Balachandran
Maher Elaissi
Md. Rashidul Hasan Masum
Mohammed Hussein Othman

Proofreaders

Simran Bhogal
Stephen Copestake
Maria Gould
Ameesha Green
Paul Hindle
Joel T. Johnson
Jonathan Todd

Commissioning Editor

Usha Iyer

Acquisition Editor

Harsha Bharwani

Indexers

Mariamammal Chettiyar
Monica Ajmera Mehta
Tejal Soni

Content Development Editor

Madhuja Chaudhari

Graphics

Disha Haria

Technical Editors

Veronica Fernandes
Pramod Kumavat

Production Coordinators

Kyle Albuquerque
Melwyn D'sa
Saiprasad Kadam

Copy Editors

Roshni Banerjee
Sarang Chari
Gladson Monteiro
Adithi Shetty

Cover Work

Melwyn D'sa

About the Authors

Mark Safronov is a professional web application developer from the Russian Federation, with experience and interest in a wide range of programming languages and technologies. He has built and participated in building different types of web applications, from pure computational ones to full-blown e-commerce sites. He is also a proponent of following the current best practices of test-first development and clean and maintainable code.

He is currently employed at Clevertch and is working on Yii-based PHP web applications. He was also a maintainer of the popular YiiBooster open source extension for some time.

Back in 2008, he translated the book *Visual Prolog 7.1 for Tyros*, Eduardo Costa, in Russian with a totally new color layout. In 2013, along with Jacob Mumm, he co-authored the book *Instant Yii Application Development Starter*, Packt Publishing.

Jeffrey Winesett is a partner at SeeSaw Labs in Austin, Texas, and has over 10 years of experience building large-scale, web-based applications. He is 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.

About the Reviewers

Ajay Balachandran is a hardcore PHP developer and an avid Yii lover from India. He is a huge advocate of writing modular, reusable, and standards-based code, which leads to his love for the Yii framework.

He is an expert in federated authentication using OpenID Connect, and now specializes in providing single sign-on and analytics solutions for the enterprise customers on behalf of his company, HiFX IT & Media Services.

Having a UI/UX background, Yii and its robust Web 2.0 oriented development has enabled Ajay to easily write applications ranging from simple shopping carts to robust APIs.

Maher Elaissi is a web developer based in Canada. He has good knowledge of object-oriented analysis and designs and specializes in PHP programming. His first experience with the Yii framework was in 2012, with a startup company Cisha GmbH based in Germany, to create an online chess game (www.chess24.com).

I would like to thank the Super Mario team (dev team) for all their support and help in producing this book.

Md. Rashidul Hasan Masum is a professional Software Engineer. Over the last 6 years, he has designed and developed a wide range of desktop and web applications using the enterprise framework Spring.NET NHibernate and websites using HTML, DHTML, JavaScript, jQuery, SignalR, Ext JS 4, ASP.NET (C#), PHP (Yii framework), Spring.NET, NHibernate, Google App Engine (Java), OpenLayer, Android with MSSQL, MySQL, and Bigtable, including sites for startup companies and small businesses. His core competency lies in complete end-to-end management of a new application development.

He also has experience in the following areas: OOP, AOP, DI, ORM, SOA, n-Tire, highly configurable applications, neural networks, and software design and testing.

He now works at OnnoRokom Software Ltd. as a Software Architect. From the beginning, they have been using the Yii framework for their large-scale web application development. S. M. Quamruzzaman Rahmani (www.byronbd.com), Project Manager, and GM Nazmul Hossain, (www.gmnazmul.com), System Analyst, have been working with him. The three of them are a super combination for teamwork according to their personality profiles.

I'd like to thank Venitha Cutinho and Akash Poojary for their coordination. Also, I'd like to thank my friend Maruf Maniruzzaman who works at Microsoft. He has taught me a lot about computer engineering. Thank you to my friend K. M. Masum Habib. I'd also like to thank Packt Publishing. I have read lots of e-books published by Packt Publishing.

Mohammed Hussein Othman is a Software Engineer who has graduated from Damascus University in Syria. He has 4 years of experience in working with the Yii framework in a variety of small and enterprise projects. Mohammed has also been working on various modern web technologies, such as PHP, ASP.NET, Ruby on Rails, Node.js, and many others. Currently, he works as a Senior Web Developer and Project Manager at Flex Solutions, which specializes in enterprise web applications.

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	1
Chapter 1: Getting Started	7
A basic application	7
Installation of a basic application template	7
Specifics of the basic application template	9
An advanced application	10
Installation of an advanced application template	11
Specifics of the advanced application template	13
Summary	14
Chapter 2: Making a Custom Application with Yii 2	15
The design stage	16
Task at hand	16
Domain model design	16
Target feature	18
Initial preparations	19
Setting up project management	19
Setting up the testing harness	20
Setting up the deployment pipeline	22
Making a web application entry point visible	26
Introducing the Yii framework into our application	26
First end-to-end test	27
Yii 2 installation to the bare code base	34
Checking the requirements	34
An introduction to Yii conventions	34
Building the wireframe code	36
Adding a controller	37
Handling possible errors	38

Making the data and application layers	39
Defining the customer model at the data layer	40
Setting up the database	41
Object-relation mapping in Yii	45
Decoupling from ORM	47
Creating the user interface	49
The Add New Customer UI	50
Routing 101	51
Layouts	52
Finishing the Add New Customer UI	53
Widgets	55
The List Customers UI	56
Customer Query UI	58
Using the application	58
Summary	60
Chapter 3: Automatically Generating the CRUD Code	63
Definition of the model to work with	63
Using Gii	64
Installing Gii into the application	64
Generating the code for the Model class	66
Generating the CRUD code	69
Finishing touches	72
Creating a new layout to support pages generated by Gii	72
An overview of the generated CRUD UI	74
Pros and cons of generated classes over manually created ones	78
Summary	78
Chapter 4: The Renderer	79
Anatomy of Yii rendering	79
The Yii application components	81
The View component	84
Algorithm to find the view files	84
Algorithm to search the layout file to be used	87
The internal workings of rendering the view file	89
Custom renderers	90
A custom response formatter	95
The asset bundles	100
An asset bundle with files from an arbitrary folder	100
Asset publishing	101
An asset bundle with files from a web-accessible folder	103

Registering CSS and JavaScript files manually	104
Placing JavaScript in different positions in the asset bundles	105
Making a custom asset bundle for our application	106
Themes	107
Making a custom snowy theme	108
Widgets	111
Summary	111
Chapter 5: User Authentication	113
Anatomy of the user login in Yii	114
Password-based login mechanics in general	114
Making the user management interface	116
Acceptance tests for the user management interface	116
Database table to store user records	118
Generating the model and CRUD code by Gii	118
Removing the password field from the autogenerated code	119
Hashing a password upon saving a user record	119
Functional tests for password hashing	120
Password hashing implementation inside the active record	123
Making a user record into an identity	126
Making the login interface	128
Specifications of user authentication	129
Making the authentication indicator	131
The login form functionality	132
The logout functionality and wrapping things up	137
Summary	138
Chapter 6: User Authorization and Access Control	139
Access control using the state of user authentication	140
FEATURE – hook methods of the controller	140
FEATURE – exception handling in Yii	143
FEATURE – controller action filters	147
Role-based access control	151
Protecting the CRM management from CRM users	152
Installing predefined users	153
RBAC managers in Yii	155
The failing test for our role hierarchy	157
Setting up the role hierarchy	159
The failing test for access control in controllers	162
FEATURE – access control filter	165
Applying access control to the site	166
Summary	170

Chapter 7: Modules	173
FEATURE – Yii modules	173
The informal concept of reachability	175
Exploring the intricacies of module configuration through simple examples	176
The Debug module	180
Building the API module	183
Building a test suite to support testing the API module	184
Defining the requirements for automatic tests of API modules	187
Moving the controller actions to a separate module	191
Retrospective on the modules mentioned in previous chapters	193
Summary	195
Chapter 8: Overall Behavior	197
FEATURE – message log	197
Actually storing the log messages	200
Setting up the e-mailing component so that log messages can be mailed	201
Reading the stored log messages	203
FEATURE – profiling	206
Error handling details	211
FEATURE – error handling controller action	213
List of built-in exceptions	214
Caching	216
FEATURE – cache component	217
FEATURE – database queries caching	220
FEATURE – page fragment caching	221
FEATURE – whole page caching	221
FEATURE – caching the request by HTTP headers	222
Minimizing the assets	223
Summary	232
Chapter 9: Making an Extension	233
Extension idea	233
Creating the extension contents	234
Preparing the boilerplate code for the extension	235
FEATURE – bootstrapping	235
FEATURE – extension registering	237
Making the bootstrap for our extension – hideous attachment of a controller	238
Making the extension installable as... erm, extension	240
Preparing the correct composer.json manifest	242

Configuring the repositories	244
Summary	251
Chapter 10: Events and Behaviors	253
Automatically marking database records with the timestamp and user ID	253
Test case for customer creation	254
Test case for updating customer updates	257
Preparing the database fields	259
Using the timestamp and blameable behaviors	260
FEATURE – behaviors	263
FEATURE – events	265
Built-in events	270
Events of yii\base\Application	271
Events of yii\base\Controller	271
Events of yii\base\Module	272
Events of yii\base\View	273
Events of yii\web\View	274
Events of yii\base\Model	274
Events of yii\db\BaseActiveRecord	275
Events of yii\db\Connection	276
Events of yii\web\Response	277
Events of yii\web\User	277
Events of yii\mail\BaseMailer	278
Summary	278
Chapter 11: The Grid	279
Dismissing of the domain layer	279
Designing for the customers' index	280
Making address, e-mail, and phone active records	281
Making the common base controller for submodels	285
Making relations from customer to address, e-mail address, and phone	288
FEATURE – widgets	290
Creating the index page for customers	292
Making a base GridView for customers	293
Changing the format of the column content	295
FEATURE – formatter	296
Making the custom GridView column for the customer audit info	299
Compressing submodels related to customers into single columns	307
FEATURE – GridView columns	308
Implementing filtering inside GridView of customers	310

Implementing sorting inside GridView of customers	316
Summary	321
Chapter 12: Route Management	323
Yii 2 routing 102	323
FEATURE – routing using names of modules, controllers, and actions	325
Fundamental rules of URL management in Yii 2	326
FEATURE – creating URLs in Yii	327
Custom routes using a configuration	328
FEATURE – URL rules	328
Custom routes using custom URL rule classes	331
Summary	334
Chapter 13: Collaborative Work	335
Configuration construction	336
Adding local overrides to the configuration	337
Console application	341
Custom console commands	342
Database migrations	345
Making custom templates for database migrations	349
Summary	352
Appendix A: Deployment Setup with Vagrant	353
Planning	354
Initial setup	354
Fine-tuning the virtual machine	356
Preparing the guest OS	356
Preparing the database and web server	357
Preparing the application	358
Using the virtual machine as a local deploy target	359
Appendix B: The Active Form Primer	361
Making the Edit form for customer	361
Active query	362
Customizing the autogenerated form	364
Passing the customer ID to submodels	371
Returning to the Update Customer form after updating the submodel	373
Custom column value for the addresses table	374
Index	377

Preface

This book is a guide that describes the process of incremental, test-first development of a web application using Yii framework Version 2.

The Yii framework, hosted at <http://www.yiiframework.com/>, is a PHP-based application framework built around the Model-View-Controller composite pattern. It is suitable for building both web and console applications, but its feature set makes it most useful for web applications. It has several code generation facilities, including the full create-read-update-delete (CRUD) interface maker. It relies heavily on the conventions expressed in its default configuration settings.

Overall, if all you need is a fancy interface for the underlying database, there is probably nothing better for you than the Yii framework. Given the extensive configuration options, any kind of application can be made from Yii.

Version 2 of the Yii framework is built utilizing the latest improvements in the PHP infrastructure collected over the years. It uses the Composer utility (see <https://getcomposer.org/>) as a primary distribution method, PSR levels 1, 2, and 4 from the PHP Framework Interop Group guidelines (see <http://www.php-fig.org/>), and PHP 5.4+ features, such as short array syntax and closures.

At the time of writing, Yii 2 is at the stage of late beta. Some changes are expected, but there should not be backward-compatibility-breaking changes anymore. Thus, the content of this book can be reasonably trusted even if the framework can attain some additional features after this book is published.

What this book covers

Chapter 1, Getting Started, covers the simplest possible methods to raise a working web application completely from scratch using the Yii framework.

Chapter 2, Making a Custom Application with Yii 2, shows how the process of implementation of a web application with a single, working, tested feature can be done from scratch using the Yii framework.

Chapter 3, Automatically Generating the CRUD Code, shows how we can implement a working, tested feature in an existing web application using only the code generation facilities and not a line of custom code written.

Chapter 4, The Renderer, describes the details of how the framework renders its output and presents some tricks to introduce customizations to the rendering process.

Chapter 5, User Authentication, discusses the tools to provide authentication to application visitors.

Chapter 6, User Authorization and Access Control, explains the ways to control access for application visitors, and, especially, about the role-based access control system.

Chapter 7, Modules, returns from the exact features of the framework to its fundamentals. Here we will clearly understand the internal structure and logic of the Yii-based application and how it influences the overall design.

Chapter 8, Overall Behavior, is about the infrastructure of the Yii-based application. We will learn about several features that affect the application as a whole.

Chapter 9, Making an Extension, tells us how to make the extension to the Yii 2 framework and prepare it so that it is installable in the same way as the extensions built-in to the basic distribution of the framework itself.

Chapter 10, Events and Behaviors, investigates the intricacies of the system inside the Yii 2 framework allowing us to attach custom behavior to many of the usual activities of the application, such as fetching a record from the database or rendering a view file.

Chapter 11, The Grid, has two purposes. First, it explains the powerful and complex GridView widget, which allows you to make complicated, table-based interfaces relatively easily. Second, it presents a *different approach* in developing applications using the Yii 2 framework, the one that is customary in its community, so you can see both the advantages and the disadvantages of both approaches.

Chapter 12, Route Management, explains the top level of the framework, that is, how it responds to HTTP requests from actual visitors.

Chapter 13, Collaborative Work, concludes the book by presenting the methods that help to manage the code base of a Yii-based application when there are several developers working on it.

Appendix A, Deployment Setup with Vagrant, shows a simple way to construct a virtual machine for your local development, which you can use for building the examples from this book.

Appendix B, The Active Form Primer, contains an extension to *Chapter 11, The Grid*, in which we use another powerful user interface widget of Yii 2, the ActiveForm. It was excluded from the chapter text because it's not directly related to the GridView widget, but we could not gloss over it completely. Without the ActiveForm, the feature we were building in *Chapter 11, The Grid*, is not complete.

Through the course of the book, starting from *Chapter 2, Making a Custom Application with Yii 2*, we'll be working with a single code base. Later chapters will build over the work previously done, so the book is expected to be read sequentially, without skipping or changing order.

Who this book is for

The text is targeted at existing, established developers who want to learn quickly whether the Yii framework can fit their demands and, especially, workflow. It is not a reference, but rather a guide. More than that, the reader is expected to have a copy of the source code and an official documentation as supplementary material while reading.

We will expect some relatively high qualifications from the reader, as several basic development concepts such as POSIX-compatible command line, version control system, deployment pipeline, automated testing harness, and an ability to navigate through the code base by fully qualified names of classes are assumed as obvious and not requiring any explanation.

What you need for this book

A workstation with the full LAMP stack installed, that is, having Apache web server, MySQL relational database management system, and PHP runtime installed over some Linux-based distribution. If the reader is capable enough, then any of these requirements can be swapped for different vendors, except PHP, which is quite obvious.

You have to use PHP Version 5.4 and higher, because it's a requirement for Yii 2, and generally, you don't have any reason to use previous versions anymore.

Even if you don't use a POSIX-compatible OS, such as any Linux-based distribution or Mac OS X, you should have a Bash-like shell, as all command-line examples in this book assume the availability of this shell.

An Internet connection is required to download many necessary libraries used throughout this book. Even if you don't update anything, you'll download approximately 320 MB of libraries, so mobile Internet probably will not cut it.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Now run the following command to create a subdirectory called `basic`, and fill it with the basic application template."

A block of code is set as follows:

```
require_once(__DIR__ . '/../../vendor/yiisoft/yii2/Yii.php');
new yii\web\Application(
    require(__DIR__ . '/../../config/web.php')
);
```


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


```
require_once(__DIR__ . '/../../vendor/yiisoft/yii2/Yii.php');
new yii\web\Application(
    require(__DIR__ . '/../../config/web.php')
);
```

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

```
$ php composer.phar require --prefer-dist yiisoft/yii2-debug "**"
```

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: "You should fill the available fields as shown in the following table, and hit the **Preview** button."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

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

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

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

Customer support

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

Downloading the example code

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

Errata

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

Piracy

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

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

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

Questions

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

1

Getting Started

Let's see how we can set up a website with Yii 2 from scratch and with a minimum amount of effort. The goal is to learn about the installation procedure of the Yii application boilerplates offered by developers and the starting set of features provided in them.

A basic application

The most basic and straightforward way to get started with Yii 2 is to use the application boilerplate published by the Yii 2 team on their GitHub repository (<https://github.com/yiisoft/yii2>) and available through the Composer tool. In the prior versions of Yii, you had to manually download and extract the archive with the framework contents. While you can still do the same in Yii 2, this version is carefully crafted so that it's especially simple to install it using the Composer tool.

Installation of a basic application template

Find a suitable directory in your hard drive and fetch the Composer **PHP archive (PHAR)** into it in any way suitable for you, for example, using the following command:

```
$ curl -sS https://getcomposer.org/installer | php
```

Now run the following command to create a subdirectory called `basic`, and fill it with the basic application template:

```
$ php composer.phar create-project --prefer-dist --stability=dev \
yiisoft/yii2-app-basic basic
```



Please note that Yii 2 specifies several system-wide dependencies for itself. You probably will need to consult the `composer.json` file inside their GitHub repository to learn about them beforehand (<https://github.com/yiisoft/yii2>). But in any case, Composer will tell you what you need to install to make Yii 2 workable. Yii 2 gets new updates quite often and its requirements are a moving target.

It's a line split into two lines for readability, with a slash denoting the overflow of the command line to the next line of text. Shell interpreters on Unix-like systems should understand this convention, so you can probably just copy and paste the code verbatim and it'll be executed correctly. You better check the documentation of Composer for the meaning of the beginning of the preceding command, but the part relevant to us is `yiisoft/yii2-app-basic basic`, which means "copy contents of the repo published at <https://github.com/yiisoft/yii2-app-basic> to our local folder named `basic`." The command will install the project skeleton in the form of a set of predefined folders, and among them the `vendor` subdirectory, which contains quite a lot of other Composer packages. Inside the `basic` folder will be your application root.

After Composer finishes installing the required packages, you can just issue the following command:

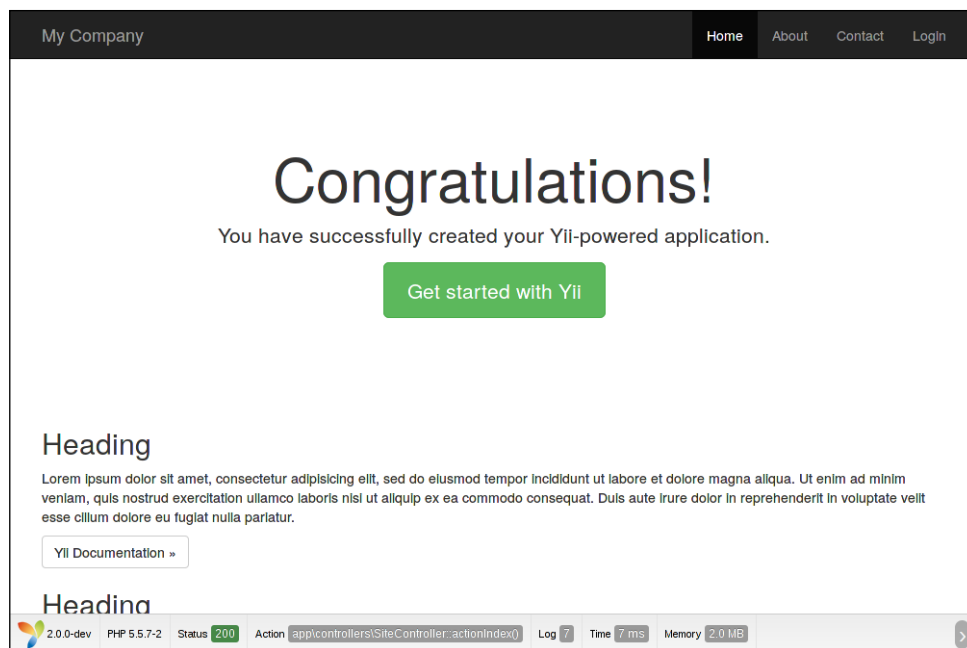
```
$ php -S localhost:8000 -t basic/web
```

Here, `8000` is the port number, which you can change to anything you want. This will launch the web server built into PHP.



This is not the preferred setup for PHP-based web applications, of course. The built-in web server was used only as a "smoke test" to verify that everything in general works. It is suitable only for local development without a heavy load. The next chapter will deal with the real-world deployment of a Yii-based web application.

Point your web browser at the `http://localhost:8000/` URL. You should see the welcome page for the Yii 2 application, which means that you're done setting things up.



Specifics of the basic application template

You can get a comprehensive overview of the various folders inside the basic template by reading the README file provided with the template (<https://github.com/yiisoft/yii2/blob/master/apps/basic/README.md>), or by reading the global Yii 2 documentation page describing basic applications (<http://www.yiiframework.com/doc-2.0/guide-start-installation.html>).

The most important thing you should understand is that the publicly available web root directory is just one folder in the overall code base. It's the web directory for our basic application. Every other folder is outside the web root directory, that is, out of reach for the web server.

As you have already seen in the installation description, given that you have PHP and, optionally, curl, this code base is ready to use right from the start; no specific environment is needed to be set up. All the dependencies are managed through the Composer tool.


A three-tier automatic testing harness is already set up in the basic template. It contains acceptance, functional, and unit tests covering most of the functionalities. The tests already included in the template are useful as examples that show how to utilize the testing framework used, which is Codeception (<http://codeception.com/>).

The template can be really useful to you if all you need is something like a news feed feature or a web tool spanning a couple of pages. However, the absence of separation by subsystems, such as the administrative backend and public frontend, will start to hinder you on a larger web application; probably, an application with more than just 10 unique routes.

Note that by reading the project dependencies in the `composer.json` file, Yii 2 has several important parts separated out as pluggable packages, and they are already included in your code base by Composer. These packages are listed as follows:


- Gii, the code generator, which we will discuss in detail in *Chapter 3, Automatically Generating the CRUD Code*
- The debug console that is already enabled on the basic application template
- A wrapper around the Codeception testing framework
- A wrapper around the SwiftMailer library (<http://swiftmailer.org/>), which can be found at <https://github.com/yiisoft/yii2-swiftmailer>
- The Twitter Bootstrap UI library packaged as a Yii 2 asset bundle (it's practically ubiquitous nowadays, but here's the link anyway: <http://getbootstrap.com/>)

The first three are set up so that you get them only when you are developing the application, since they are useless and even harmful in the production environment. Most probably, you'll need all of these on any serious project though.



A short overview of the basic application installation:

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar create-project --prefer-dist \
--stability=dev yiisoft/yii2-app-basic basic
$ php -S localhost:8000 -t basic/web
```



An advanced application

Apart from the basic application template, Yii 2 has an advanced application template. It's geared more towards medium-sized applications (such as applications that are really useful to businesses), and its main feature is two separate web interfaces: one dedicated to content management and the other to presenting this content to visitors. So, you get an almost complete CMS skeleton with this template.

Installation of an advanced application template

The first steps are the same as for a basic template. You need to fetch the Composer executable and set up a new project using it:

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar create-project --prefer-dist --stability=dev \
yiisoft/yii2-app-advanced advanced
```

You can see that the difference is just that instead of the word "basic", we use the word "advanced".

Now, let's make further changes. First, go to the newly created directory named `advanced`. After this, you need to generate the required local configuration by running the following command:

```
$ ./init
```

Yes, it's just the `init` script from the root of the code base. It'll ask you whether you want a development mode or a production mode and create all the necessary auxiliary configuration snippets and entry scripts. To be precise, it just copies the contents of the `dev` or `prod` folders from the `environments` subdirectory depending on whether you selected the development environment or the production one. Just open the `environments` subdirectory and you'll understand how it works.

Next, you need to create the database to be used by this application. By default, for configuration of a development environment, you have to set up a MySQL database named `yii2advanced` accessible from the `localhost` at the default MySQL port for user `root` without any password. You can see the details in the `common/config/main-local.php` file.

Given that you have the database set up, you need to run migrations. We'll talk about migration scripts in the next chapter (and we will even write several scripts ourselves), but if the very concept of **database migrations** is foreign to you, you can read about it in the official documentation at the Yii 2 website (at the time of writing this, there is no official documentation, but the framework has the docs included in the GitHub repository at <https://github.com/yiisoft/yii2/blob/master/docs/guide/db-migrations.md>).

Just run the following command anyway:

```
$ ./yii migrate
```

It'll present you with a list of exactly one migration and ask you for confirmation before doing its job.

Now, you are ready. Make both the sides of the application accessible for you by executing the following commands:

```
$ php -S localhost:8080 -t frontend/web
$ php -S localhost:8081 -t backend/web
```

As with the basic application template, we are using the built-in PHP web server just because it's a lot simpler to be demonstrated in the book than explain how to set up Apache or some other web server to serve from these folders.

Now you'll have the backend side of the application intended to be used by content managers and the frontend side of the application intended to be the website your visitors will see. Also, note that you have a completely controllable console runner launched by the `yii` script you used when doing migrations.

Advanced application has exactly the same frontend as basic application. Here is how its backend looks like:

The screenshot shows a web application interface. At the top, there is a dark header bar with the text 'My Company' on the left and 'Home' and 'Login' on the right. Below the header, there is a light gray navigation bar with 'Home' and 'Login' links. The main content area is titled 'Login' and contains the text 'Please fill out the following fields to login:'. There are two input fields: 'Username' and 'Password'. Below the 'Password' field is a checkbox labeled 'Remember Me'. A blue 'Login' button is positioned below the checkbox. At the bottom of the page, there is a light gray footer bar with '© My Company 2014' on the left and 'Powered by Yii Framework' on the right.

The advanced template backend is locked down initially. After logging in, you get the same page as the one from the basic template or the advanced template frontend, but with only the login feature in the menu.

Specifics of the advanced application template

The most important thing about the advanced application template is that it is three *basic* application templates wired together as one:

- Inside the `frontend` folder is the application structure for the public-facing side of your website. Real functionalities and the content of your website or web application is expected to be placed here.
- The folder named `backend` is for your CMS, protected from unauthorized access. You are expected to place all of your admin-accessible CRUD here.
- The `console` folder is mainly for your custom console commands, in hope you'll have any, and a lot more likely, for your migration scripts.
- The `common` folder contains code that will be used by all the entry points, since it's a single application after all.

Of course, nobody forces you to use the frontend and backend sides as described. You just have two web frontends sharing the same code base, so you are free to use them as you wish. However, the UI already prepared for the advanced template that has a password-protected backend right from the start.

You should know about the login feature for the freshly installed advanced application template. Initially, it had no users defined, and you had to create one by utilizing the signup feature at the frontend. After that, you'll be able to login to both backend and frontend using the created credentials. The frontend is identical to the basic application, and the backend is stripped of everything except the login feature and front page, so everything is up to you.



A short overview of the installation of the advanced application:

```
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar create-project --prefer-dist \
--stability=dev yiisoft/yii2-app-advanced advanced
$ cd advanced
$ ./init
$ mysql -u root -e 'create database yii2advanced'
$ ./yii migrate
$ php -S localhost:8080 -t frontend/web
$ php -S localhost:8081 -t backend/web
```

Summary

Yii 2 allows you to configure almost all paths used by the framework, and hence you can create any directory tree you wish. By utilizing the PHP 5.3 namespaces wisely, you can even have a physical structure of your project different from the logical one, that is, your files will lie in folders differently from how your classes are structured by namespaces. This will surely be quite tedious to do though.

In the next chapter, we'll look at how we can utilize Yii in a (albeit small) real-world project, built completely from scratch, and without using the templates we saw in this chapter.

2

Making a Custom Application with Yii 2

In this chapter, we'll see how Yii can help us build web applications. The example will be reasonably small, but it will be done using proper software engineering disciplines. We'll go through all the steps of application development, each step backed by the bleeding-edge best practice described in the definitive books on this topic:

- **Building the domain model:** This is explained in *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Eric Evans, Addison-Wesley Professional
- **Setting up the testing harness:** We'll follow the acceptance test-driven development practice described in *Growing Object-oriented Software, Guided by Tests*, Steve Freeman and Nat Pryce, Addison-Wesley Professional
- **Setting up the deployment pipeline:** This is explained in the following books:
 - *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Jez Humble and David Farley, Addison-Wesley Professional
 - *Continuous Integration: Improving Software Quality and Reducing Risk*, Paul M. Duvall, Steve Matyas, and Andrew Glover, Addison-Wesley Professional

- **The Red-Green-Refactor development cycle:** This is explained in depth in the following books:
 - *Clean Code: A Handbook of Agile Software Craftsmanship*, Robert Martin, Prentice Hall
 - *Test-Driven Development by Example*, Kent Beck, Addison-Wesley Professional
- **Deployment and manual tests:** This fits into the *Continuous Delivery* paradigm too, but these steps are inevitable anyway

Stay focused.

The design stage

Pay attention to the fact that we will be using this example application through the whole book. In this section we'll define the landscape for the whole adventure before us.

Task at hand

Let's pretend we are a small business providing some services. We have a particular number of clients we have connections with, and the amount is so large that managing it on paper and business cards is too unwieldy. So, we need some sort of automated way of finding the full profile about a given client.

For starters, we need some sort of create-read-update-delete (**CRUD**) user interface for simple records describing the most essential attributes of our clients.

It's obvious that as we as a business will grow and evolve, the same will happen with our client management, and so our application will grow and evolve too. We should account for changes right from the start.

As we will be eating our own dog food, this software better be of the highest possible quality.

Domain model design

Obviously, we will be dealing with customer models in our application. Between the "customer" and "client" terms, we choose "customer" for accuracy.

A customer is a person who has a name, address, e-mail, and phone number at the minimum. We provide services for customers, which are counted in hours, and we are being paid fees for these hours according to the contract. That's what we will include in the first iteration of designing.

Our primary assumption is that each customer is a single person, so we don't deal with companies that can have multiple people as contacts. The name is an incredibly complex construct; if we are to delve into the details of its structure, such as honorifics, titles, nicknames, middle names, patronymics, and so on. But we aren't really interested in the structure of the customer's name, we need it only for identification purposes. So, we will represent it as a line of text, allowing us to write a name in free format. The address is a construct of the same complexity, but this time we have to retain the structure instead of using plain strings again, because we will need to do two things with addresses:

- Calculate some statistics, such as how many customers we have in a particular city
- Properly generate the address lines according to the postal rules in different cultures

So, we decide on the structure as follows:

- Purpose (for example, billing address, shipping address, home address, or work address)
- Country
- State, for countries partitioned into several regions, such as the USA
- City
- Street
- Building
- Apartment/office
- Receiver name
- Postal code

We should note that an address can be for an apartment, postal box, office in an office building, employee in an organization, or for a whole building. Also, a customer can have many addresses.

The phone entity has the following attributes:

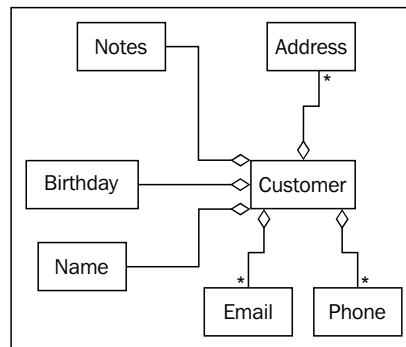
- Purpose (personal or work)
- Number

A customer can have several phone numbers, hence the "purpose" field.

Apart from names, addresses, and phone entities, our staff will surely need a way to assign a free-form textual description of a customer, which we'll simply name notes. Also, taking note of a birthday would be cool too. And e-mail, of course. By the way, a single customer can have several e-mails.

Let's stop here.

We can now figure out the complete aggregate for the customer model, which is depicted in the following diagram:



According to the *Eric Evans* book *Domain-Driven Design*, the customer is an entity, that is, an object whose state will surely change over time, and we care about its identity across the system. Everything else is a value object, that is, an object whose state will not change after initial creation, and thus they are completely interchangeable.

For the sake of simplicity, we will not detail how business is done with the customer, because we will just not be able to cover it all across this book. However, let's mention that we have some sort of services we provide to our customers, and it'll be useful to maintain records of them too. This model will be used in the following chapter.

Target feature

Let's fulfill one specific task. Given that someone called us by phone (assuming we have identified the number), we want to get all the details we have gathered so far about the person who's calling. If we don't have such a number associated with someone registered in our application, then we know he or she is not our customer (yet). If we do have the number registered, then we can at least greet this person by his or her name, which is superb customer service.

We should understand that to make queries to the database, we need a way to at least insert data into it and potentially a way to edit and delete it. Thus, our feature set for the first iteration of development will be as follows:

- To record info about a customer into the database
- To edit info about a customer in the database
- To delete info about a customer from the database
- To query info about a customer by his or her phone number from the database

Building the way to make generic queries to a database is not the goal. We will only deal with the queries containing a phone number.

Let's begin then.

Initial preparations

We will be working on the same application in all of the following chapters until the end of the book, so the preparations are to be done only once.



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.

Setting up project management

Our example application is essentially a Customer Relation Management one, that is, a CRM application. Thus, we will start with a folder named `crmapp`.



Please note that all examples of command-line invocations *through the whole book* assume that your current working directory is the `crmapp` folder and not anywhere deeper or higher in the filesystem hierarchy.

Version 2 of Yii has the Composer package manager as the preferred method of installation, so we will use this tool too. While you can read the details in Composer's full documentation, here's a crash course of it:

- All packages installed by Composer are kept inside a special subdirectory under your project's root called `vendor`.
- All dependencies and other information about your data relevant to Composer are kept in the manifest file called `composer.json` under your project's root. As long as you have the dependencies declared there, you can safely purge the `vendor` folder at any time as it'll be repopulated by the next call to `php composer.phar install` or `php composer.phar update`.

The documentation for Composer presents a nice one-line command that will get the Composer executable to you:

```
$ curl -sS https://getcomposer.org/installer | php
```


Of course, if you don't have CURL lying somewhere in your PATH, you can just go to the official site of Composer at <https://getcomposer.org/> and grab the PHAR archive from there.

After that, Composer will be invoked whenever `composer.phar` is called:

```
$ php composer.phar <command>
```

It is assumed that you use some version control system for your code base. The code bundle for this book uses Git (<http://git-scm.com/>).

[



```
Preparations in essence
$ mkdir crmapp
$ cd crmapp
$ curl -sS https://getcomposer.org/installer | php
$ git init
```

]

Setting up the testing harness

As we declared in the opening paragraph of this chapter, we will follow the test-first development practice based on acceptance tests. The reasons behind this decision are as follows:

- We will have a way to check whether the application works as intended without resorting to tedious manual testing
- We do not have a real need for deep unit testing because most of our work will involve wiring together already-existing components with rudimentary logic in between, so end-to-end acceptance tests through the UI is the most simple and viable solution


We need some form of acceptance tests in the system anyway if we care about user feature requests at all.

Yii 2 has support for the **Codeception** testing framework built-in, with <http://codeception.com/> being the official website for it. We will never use it in this book, but the extension named `yii2-codeception` (see <https://github.com/yiisoft/yii2-codeception>) provides a set of helper classes to integrate your tests more deeply with the Yii framework.

Let's declare that we want Codeception available in our project. Run the following command:

```
$ php composer.phar require "codeception/codeception:~2.0"
```

Wait a bit until Composer finishes.




Here are the contents of your `composer.json` at this point:

```
{
    "require": {
        "codeception/codeception": "~2.0",
    }
}
```

The `php composer.phar require <packagename:version>` command is just a helper method to insert lines inside the `require` block of the manifest and call the update routine.

Of course, we'll need to add Yii 2 as a dependency at some point, but for now, let's do one thing at a time.



Now, we have the Codeception executable available at `./vendor/bin/codecept`. This location is a bit long to type, so a POSIX-compatible shell like `bash` allows you to reduce it as follows:

```
$ alias cept="./vendor/bin/codecept"
```

It's better now. In all of the following command-line examples in this chapter, we assume you have done this substitution.

Codeception is a complex system, so we'll need to rely on its self-building system for now. To not delve unnecessarily into the inner workings of Codeception, let's just stick to defaults. Call the following command:

```
$ cept bootstrap
```

This will generate a `tests` directory and configuration tree for Codeception.

Now, let's create the first dummy acceptance test to check the top level of our test harness:

```
$ cept generate:cept acceptance SmokeTest
```

This command will generate `SmokeTestCept.php` in the `tests/acceptance` directory. When you open it, you'll see something like the following, depending on the version of Codeception:

```
$I = new AcceptanceTester($scenario);
$I->wantTo('perform actions and see result');
```

`AcceptanceTester` is the class of objects holding all the methods we can use to test our application imitating a real user behind the browser. Codeception also has `CodeGuy` for unit tests and `TestGuy` for functional tests, but that's for later.

When we say `AcceptanceTester.wantTo("do something")`, we just create a title (enclosed in double quotes) for the test actions following this invocation.

Let's change the dummy test to a simple smoke test that our landing page is up for:

```
$I = new AcceptanceTester($scenario);
$I->wantTo('See that landing page is up');
$I->amOnPage('/');
$I->see('Our CRM');
```

So, we expect to see the line **Our CRM** when we access the landing page of our future application. Let's pretend that we'll have such a title somewhere in there.

Now we run the test:

```
$ cept run
```

We see it fail because we don't have the web server serving anything on the `/` request. Thus, we arrive at the point where we need to write production code to satisfy our tests. However, right now, it's not the production code we need, but the infrastructure to serve it. We need a machine to deploy to.

Setting up the deployment pipeline

The problem is described here. The web acceptance tests that we will be writing imitate a real user who opens the web application in the browser and interacts with it using the visible UI. So, for them to work, we need an application that is fully deployed to somewhere accessible from the machine on which we will run acceptance tests.



In most cases, you'll decide to just run the application on the same machine on which you'll do the source code editing. Wrong! Don't do it.

Most probably your own workstation is not the same as the machine your web application will ultimately run on. This has been an ongoing problem in the industry for decades now, and you can be sure that when your application's lifetime is measured in years, you will get the same integration problems if you test your application on the machine with a different environment than the production server. Of course, this doesn't apply to the prepackaged software that you sell to various users and when you require portability. In our case, we assume a stationary web application for a single deploy point, so portability is not an issue, but reproducible tests are an issue.

Ultimately, your acceptance testing will consist of the following steps:

1. Deploy the application to the test server.
2. Run acceptance tests on your machine.

Of course, you can run acceptance tests on the test server. To do so, you just need to configure tests to use the usual loopback network interface, `localhost`. However, it will require you to install additional software for your test server irrelevant to the application itself. For example, if you decide to run full-stack, in-browser tests using **Selenium**, you'll probably need to install a web browser, Java runtime, and virtual framebuffer software on the test server, and this will lead to installation of a significant amount of system-related libraries, which probably is just a waste. It's a lot more efficient to use your own desktop environment to run the web acceptance tests.



This cannot be said about unit and functional tests, of course. Unit tests, due to their nature, are run on the raw code base, without the need to deploy at all. Functional tests should be run on the deployed application because they are required to test the validity of interactions between the configured and working parts of the final application.

In any case, ideally you should end with a simple command, named something like `deploy`, which will do the following:

1. Access and launch the target machine (especially if it's a virtual machine instance).
2. Ensure that there is a valid environment expected by the application.
3. Copy the current state of the code base to the target machine.

4. Configure the copied code base for the environment on the target machine.
5. Launch the application.

You should be able to do all of the preceding steps by typing `deploy` in the command line and hitting *Enter*. As Martin Fowler said in his definitive article *Continuous Integration* (seen last time at <http://martinfowler.com/articles/continuousIntegration.html>), this should become a non-event for you. Ideally, deployment should happen automatically when you launch the acceptance test harness.

In this book, we'll concern ourselves with only the last two steps of the procedure. As we're working with a PHP application, the "launch the application" step typically will be completed as soon as we have a web server running on a target machine.

This is a book about web application development and not about system maintenance, and it's targeted at web developers and not operation engineers. However, in *Appendix A, Deployment Setup with Vagrant*, we prepared the description of one setup based on the usage of a virtual machine, which you can easily repeat on just any desktop workstation. You will not need a separate physical machine, and you will still be able to imitate a real-world deploy procedure. If you don't have other options, you are strongly encouraged to read it. In fact, all of the code in this book was prepared using the setup described there. Let's pretend that you have an environment prepared with a `deploy` command, and for simplicity, we assume it'll be run before each run of the acceptance testing suite. The result of your `deploy` should be the single URL accessible from your machine, which the acceptance testing harness will use as the entry point to your application.

Now, let's go to the section of Codeception configuration that is relevant for the acceptance test suite within the file `tests/acceptance.suite.yml`, and add that URL in the `modules.config.phpBrowser.url` token. The file, assuming you did not modify anything else and nothing has changed in the default Codeception installation since this chapter was written, should look like the following:

```
class_name: AcceptanceTester
modules:
  enabled:
    - PhpBrowser
    - WebHelper
  config:
    PhpBrowser:
      url: 'http://YOUR.APPLICATION.URL'
```

For example, if you configure the target machine with the Apache web server using the IP-based virtual host technique (as described at <https://httpd.apache.org/docs/2.2/vhosts/ip-based.html>), the `modules.config.PhpBrowser.url` value can look like `http://127.0.0.1:8000`.

As we change the configuration, we should rebuild the Codeception harness. Here is the command to do it:

```
$ cept build
```

Do not forget that `cept` is an alias we created ourselves. The real executable is in the `./vendor/bin/codecept` file.

If you run the tests now:

```
$ cept run
```

You will see an output as shown in the following screenshot:

```
0.0 [[detached from 2f2a905)*] $ vendor/codeception/codeception run acceptance
Codeception PHP Testing Framework v2.0.0-beta
Powered by PHPUnit 4.1-dev by Sebastian Bergmann.

Acceptance Tests (1) -----
Trying to See that landing page is up (SmokeTestCept.php) Fail

Time: 97 ms, Memory: 10.00Mb

There was 1 failure:
-----
1) Failed to see that landing page is up in SmokeTestCept.php (/home/hijarian/projects/crmapp/tests/acceptance/SmokeTestCept.php)
Sorry, I couldn't see "Our CRM":
Failed asserting that
--> /<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
  <head>
    <title>Index of </title>
  </head>
  <body>
    <h1>Index of </h1>
    <table>
      <tr><th valign="top"></th><th><a href="?C=N;O=D">Name</a></th><th><a href="?C=M;O=A">Last
modified</a></th><th>
[Content too long to display. See complete response in '_log' directory]
--> contains "our crm".

Scenario Steps:
2. I see "Our CRM"
1. I am on page "/"

FAILURES!
Tests: 1, Assertions: 1, Failures: 1
```

You'll see that Codeception now shows something on the `/` route but not what we expected it to. It'll either be a 404 error or 403 error, depending on the version of Apache used, or maybe something else if you are using a different web server. Anyway, the root of the problem is simple, that is, we need an `index.php` file inside the web-accessible directory.

Making a web application entry point visible

Let's decide on the convention here: the only folder that will be accessible from the Web will be called `web`, placed at the root of the code base. For example, if your web server is Apache, it'll be the `web` folder's path that you put into the `DocumentRoot` directory.

Given that, just put the following content as the `index.php` file in the `web` subdirectory:

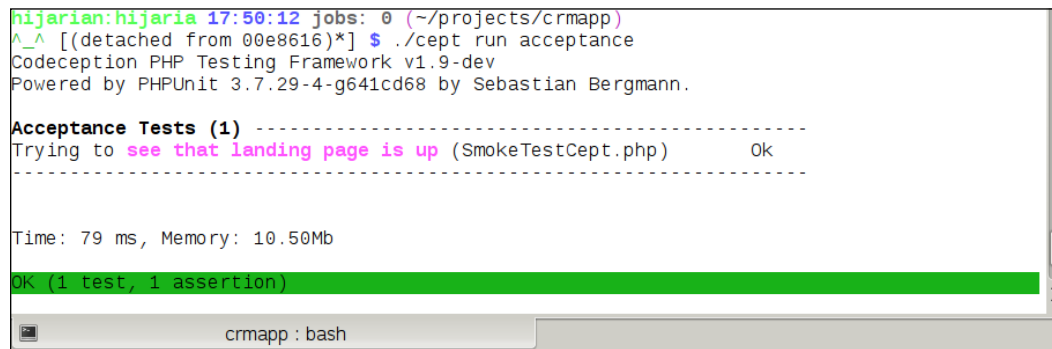
```
Our CRM
```

Yes, just a seven-character text file. After all, that's everything our acceptance test expected, right?

Then we run the tests:

```
$ cept run
```

We get the following output:

A terminal window showing the output of the 'cept run' command. The prompt is 'hijarian:hijaria 17:50:12 jobs: 0 (~/projects/crmapp)'. The command executed is '^_^ [(detached from 00e8616)*] \$./cept run acceptance'. The output shows 'Codeception PHP Testing Framework v1.9-dev' and 'Powered by PHPUnit 3.7.29-4-g641cd68 by Sebastian Bergmann.'. A section titled 'Acceptance Tests (1)' follows, with a dashed line separator. Below it, it says 'Trying to see that landing page is up (SmokeTestCept.php) Ok'. Another dashed line separator follows. Then, it shows 'Time: 79 ms, Memory: 10.50Mb'. At the bottom, a green bar highlights 'OK (1 test, 1 assertion)'. The terminal title bar at the bottom says 'crmapp : bash'.

```
hijarian:hijaria 17:50:12 jobs: 0 (~/projects/crmapp)
^_^ [(detached from 00e8616)*] $ ./cept run acceptance
Codeception PHP Testing Framework v1.9-dev
Powered by PHPUnit 3.7.29-4-g641cd68 by Sebastian Bergmann.

Acceptance Tests (1) -----
Trying to see that landing page is up (SmokeTestCept.php)      Ok
-----

Time: 79 ms, Memory: 10.50Mb

OK (1 test, 1 assertion)
```

Now we need to allow ourselves to use Yii 2 in our project. An easy way to do this is just to write the full, end-to-end test, which describes our desired functionality.

Introducing the Yii framework into our application

Now that we have the entire supporting infrastructure we need to begin working with, let's return to our first feature we defined at the design stage and define the acceptance test for it.