The No Bullshit Bible : Creating Web 2.0 Startups & Programming

Written by James Gillmore
Edited by Holly Welch

FaceySpacey
www.faceyspacey.com

Yii

TABLE OF CONTENTS

Technical

Chapter 3

3-1 Yii | INTRO TO FRAMEWORKS

Typically now, you'd be introduced some raw MySQL. The reality is to get productive you can skip to using a framework that will "abstract" all the MySQL code you'd have to type. To get advanced, you'll most definitely have to master SQL ("Standard Query Language"). But keep in mind we're on the fast track to getting productive and getting the general-to-specific understanding of the entire scope of what you need to know...By the way "abstract" means in layman's terms to make coding easier by creating a layer on top of deeper code that makes the most common tasks easier. So frameworks provide a precise toolset (i.e. set of code commands you can type) that handles the most common tasks you will need to execute without having to know all the internals. The result is you get to remember a lot less coding commands to accomplish major things.

The framework we recommend is Yii, which is the best PHP framework around right now as of August (2011). period. Don't waste your time researching all the different PHP frameworks available. Just learn and master Yii: http://www.yiiframework.com .

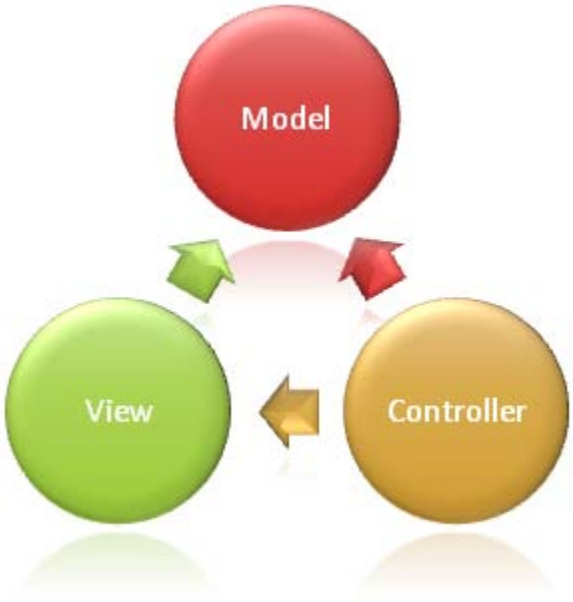
Yii is an MVC framework. That means it's based on the "*Model View Controller*" designer pattern. We'll cover each of these 3 aspects in depth in the following Yii tutorials. For now let's start with an example of the most important part, the *Model,* so you can get an idea of the power of the Yii Framework to inspire you to continue your study of it. We're going to create an Article class, which is essentially the template for *model objects* to be generated from:

```
class Article extends CActiveRecord
{

   public function tableName()
   {
      return 'article';
   }
```

```
    public function getInfo()
    {
      echo ‘The book, ‘.$this>title.‘ has ‘.
$this>pageCount.‘pages<br>’;
    }
}
```

Ok, so notice, we did not define the “*title*” and “*pageCount*” properties. The reason is because this*Article class* automatically grabs these properties from columns in your article table in the database. We’ll cover SQL later, but real quick: the idea is a database is a series of spreadsheets with column headers, rows with similar data, and cells that hold the data that correspond to the current column each are in. So the idea here is that Yii simplifies how your PHP code works with the database. It inherently knows the columns of the corresponding table, and provides them as properties for all *Article objects* that extend from the *Article class*.



How does this happen? Well, through inheritance. This *Article class* you created extends *CActiveRecord*, which is a class within the Yii Framework. That means you

have access to all the pre-built tools it provides. That's it. By the way the *Article class* is called a "*model*"--it's the "M" in "MVC." It automatically models database tables.

So it lets you do things like tack on the *getInfo()* method and not have to do anything else to start getting results.

That's the end of this lesson. The takeaway is that you create your classes by extending from classes in the Yii framework, and a whole lot of work is automatically done for you. I explain how to install Yii inServer Setup 3 - Installing Yii & Your Application, but the basic idea is you associate the application you're working on with your installation of the Yii Framework, and now you can reference classes, methods, etc, from the library of code provided in the Yii Framework.

3-2 Yii | CONTROLLERS

In this tutorial you'll learn the path of execution in your code from when a visitor first pings a URL on your site to the HTML delivered to the browser from your controllers and their actions.

Ok, so check it, remember at the beginning of the PHP tutorials you learned how web browsers connect to servers by accessing a script file remotely, and then the server executes the result and serves up the result to your browser, right? Well, in Yii and most MVC Frameworks ("Model View Controller") only one script is ever pinged. You're not pinging yoursite.com/script1.php and then yoursite.com/script2.php and then yoursite.com/contact.php, etc. Yii is wired in conjunction with some help of your server (typically Apache, which we'll cover later, and which ultimately won't be a big deal in getting productive quick) to make it so all requests to your site go through the same script, called index.php. In the browser, your Apache server will hide index.php from all URLs so you can access your site at yoursite.com without having to access yoursite.com/index.php. Another example would be yoursite.com/about which also would send a request through index.php.

What happens then is that the internals of all the Yii library code route the request to the proper set of Yii code. So if */about* is in the URL, Yii will send the request to a code class called "*about*" and that's where you write your code. Basically Yii will interepret pretty urls that don't end with "*.php*" and route the request based on the words in the complete URL to the appropriate code to execute. capiche.

Code snippet time:

```
class AboutController extends CController
{
   public function actionUs()
   {
      $aboutUsText = "Some stuff about how great your new startup
is...";
      $this->render('viewFile', array('content'=>$aboutUsText));
   }
}
```

Now if you were to visit yoursite.com/about/us this code would be triggered. Why? Well obviously because the bolded word "Us" comes after the word "action." Yii controllers

(which as you can see extend from CController) are programmed so that any method that starts with the word “action” correspond to specific pages requested. So the idea is when your request starts with “/about” then theAboutController is called, and specifically within it if after “/about” is “/us” like this “/about/us”, it’s “action method” that ends in “Us” is executed.

Now, to get a page to render to the browser, you have to call the “*render()*” method of the current controller class, which as you remember will require using the “*$this*” keyword. Yii controllers have a method called “render” which is all about displaying a specific view file, in this case called “*viewFile*” which is bolded to point out its importance. That view file (which is a php code file) is then displayed in the browser. That’s it.

Lastly, the view file (which will be named somewhere as “*viewFile.php*”) will be passed a variable called *$content* with the value of *$aboutUsText*. So in the view file you can place the *$content* variable anywhere you want to display the paragraph about your startup. The view file will be a combination of HTML and PHP snippets of dynamic code interspersed within it.

3-3 Yii | VIEWS

Views are your HTML templates that are delivered to your browser, filled with dynamic content generated by PHP. Let’s start by creating the bare minimum view necessary to show your About Us page:

```
<h1>About Us</h1>

<div>
   <?php echo $content; ?>
</div>
```

We're actually going to jump to learning HTML in the next tutorial. So I won't go too deep into explaining what *<h1></h1>* and * * means. In general, they're called "*tags*," and pairs of 2 matching tags (the ending one starting with a backslash) are called "*elements*."

The idea here is simply that we display the value of the *$content* variable (i.e. your paragraph about your startup) right there in the middle of the HTML.

You may be wondering why we don't just write that paragraph in plain text/html right there, rather than go through all the work of defining php variables in the corresponding controller class and then passing it to this view. Well, the answer is one you'll typically hear while studying programming tutorials: this is an easy example that shows the core features you need to learn, and when you want to do more complex things (specifically dynamic things, where the value of *$content* can change) the utility of coding like this comes to light. Basically we could use the above HTML view file as a template for many similar pages, such as those in a blog, and display different blog article content via the variables, rather than code a different view file template for every single blog article. I'll provide a more complex example in later Yii tutorials that highlights the power of what you can do here.

3-4 Yii | FILE STRUCTURE

Ok, so by now, you should be wondering how the controller class's *render()* method knows where the "*viewFile*" view is. The answer is simple, Yii basically requires you to organize your code files in a structured set of folders. Specifically here's what your Yii directory structure will look like:

```
>yoursite
   >controllers
      >AboutController.php
   >models
```

```
      >Article.php
   >views
      >viewFile.php
```

That's an oversimplification that those familiar with Yii may pin as incorrect, but again the point is to get the main high level concepts here, not master every precise specific. Learning it this way will make it 10 times easier to learn the specifics later. And yes, I'm going to keep repeating the technique of learning/teaching I propose throughout my FaceySpacey tutorials.

So basically the code you write in the *AboutController.php* class knows that when you want to render a view file named "*viewFile*" to render a file in the following file: */yoursite/views/viewFile.php*.

They call this "convention over configuration," which is basically a motto Yii attempts to adhere to in order to do things like keep the file structure similar from project to project--i.e. Yii projects all follow a similar file structure convention/style. In this case, this saves many lines of code where you have to define full file paths every time you want to reference another file of code.

3-5 Yii | PUTTING IT ALL TOGETHER

So let's wrap up the framework tutorial by doing something a little dynamic to highlight the power of a framework.

```
class ArticleController extends CController
{

   public function actionView()
   {
      $article = Article::model()->findByAttributes(array(
         'title'=>$_REQUEST['name']));
      $this->render('viewFile', array('model'=>$article));
```

```
    }
}
```

So what could we possibly be doing here? We're dynamically finding a specific article and rendering it to the view. The idea is this code is made to render 1 of many articles in your blog, not just the same about us paragraph. In technical terms, we're finding an article in the database, and passing its corresponding object that Yii generates to the view.

The view file would look like this:

```
<h1><?php echo $model->title; ?></h1>

<div>
   <?php echo $model->body; ?>
</div>
```

The view file here is passed the entire *$model* object, rather than just a plain variable that holds a single variable (i.e. previously the about us paragraph). This object has properties for the article title and the body of its article.

So the next question you should ask yourself is: ok so how does the blog know which blog article to display? The answer is simple: through clues in the URL you provide.

In this case, the URL will be something like this: *yoursite.com/article/view/**name/the-article-name***.

Let's break down this url:

/article *= the controller to use*

/view *= the controller's action to use*

/name *= the key in the "global" PHP array called $_REQUEST*

/the-article-name *= name of the article.*

So basically PHP has a global array called **$_REQUEST** that you can use anywhere in your code (in any scope) which stores values in the URL. In standard php, urls would be structured like this:

*yoursite.com/article/view?**name=the-article-name.***

However, in order to provide more beautiful URLs the Yii framework is programmed to give you access to that same data by structuring your urls using standard file path notation, i.e. */name/the-article-name*.

Then in the php code you can simply access *$_REQUEST['name']* to get the value "*the-article-name*";

Finally, this line:

```
Article::model()->findByAttributes(array('title'=>$_REQUEST['name']));
```

loops up an article in the article database table that has the title stored in *$_REQUEST['name']*.

So this allows your blog site to have URLs leading to different articles like this:

yoursite.com/article/view/name/article-1.

yoursite.com/article/view/name/article-2

yoursite.com/article/view/name/article-3.

Final note: you may be wondering why the title is hyphenated. In reality, your article table would have a column called “url” or “slug” which is basically the hyphenated version of the article title. You could also have URLs like this:

yoursite.com/article/view/id/1

yoursite.com/article/view/id/2

yoursite.com/article/view/id/3

And then your finder code would look up the article in the database by its unique ID number column like so:

```
Article::model()->findByPk(array('id=>$_REQUEST['id']));
```

Note: the *findByPk()* method standards for Find by Primary Key, which usually is a numerical auto-incrementing unique ID #.

The crux of what’s going on here is that the provided information in the URL can be different. Therefore your application code needs to behave dynamically and know how to deal with different provided information. This is basically the cornerstone of dynamic programming. It’s what makes coding in languages like PHP so different than the static HTML language, which is just basic commands about the placement/style of elements on a web page. In short, the code you write needs to be able to deal with various possibilities and still produce a result. Your code is expecting different “input” so it can give you varied, aka “dynamic” output.

The true true cornerstone here is this variable *$_REQUEST['name']*, which can hold different values. It’s one of the many ways your application can receive varied input.

So, fueled with a database of articles (and their urls/slugs), you have a ton of data to run searches on via Yii’s various model finder methods, i.e. *Article::model()-*

>findByPk(). I'll explain what's going on in that exact bit of code in a second, but first I'd like to finish my point here: the point is that with all the data in your database you're innately prepared to deal with all sorts of input, such as the article ID # provided in the requested URL. You can then basically make a comparison and see if any data in your database has a matching ID #, and get the corresponding database row. From there it's simple: yii turns it into a model object, and you can access the various columns in that row.

So back to this line:

```
Article::model()->findByPk()
```

All that is there is a common yii code structure to look up articles in the article table by primary key. You have to start with *Article::model()->* . Why it starts with that is beyond the scope of this article, but the basic point is that your Article class has these tools available to it, and because of its*tableName()* method which you previously saw that returns the text string "article", knows to look for rows in the article table by their ID #. Yii has a great deal of tools to perform complex database lookups. This is the easiest example.

That concludes the introduction to frameworks set of tutorials. You're now armed with a basic understanding of the goals that *MVC frameworks* accomplish. You should start imagining how your application will work to dynamically display different content based on varying input. Keep in mind there are other places users can provide input: cookies (i.e. data your site stores for a long period of time in your visitors' web browser), data submitted via forms, etc. Cookies, forms, URLs are really the main 3. They may actually be the only 3. But that won't stop you from building wonderful complex and innovative applications. That's all you need. After this initial input, a whole signal flow process can ensue in your application where one bit of code functions as input to another block of code, and then that block of code's output is the input to yet another

block of code. For example, if the user's user ID was stored in a cookie, with every request you could lookup in your user table within your database the row corresponding to the current visitor, and then put that visitor's name on every page. So the idea is you originally had a user ID as input from a cookie, and that served as input to a database lookup in the User table, and then that output served as input to a function that combines the first and last name to create a complete name, which then was output to the browser. That's a chain or flow of continuous input and ouput, and you can do as much as you want in that chain, and it can go on for as long as you want. The result will always be the same though: you pass the result as an HTML web page to the browser from your server.

Conclusion & Further Reading

We at FaceySpacey hope you've enjoyed our FaceySpacey Bible, and are coming away many times more ready to succeed at your next software startup. At the very least, you should have a birds-eye-view of what you need to do to get your startup, and have quelled a lot of insecurities you may have had regarding how you should execute it. That said, I will point to you to what you should do next.

As promised, the following is a list of the precise books I read to master web development using HTML/CSS, Javascript, PHP & MySQL. They are presented in the best order to most efficiently learn the subject at hand. It's similar to the order I read them in, but enhanced based on what I learned and the order I wish I read them in. Good luck:

HTML/CSS:

CSS Mastery: Advanced Web Standards Solutions

http://www.amazon.com/CSS-Mastery-Advanced-Standards-Solutions/dp/1430223979/

Before you start coding PHP, Javascript, etc, understand how HTML works. This is where you start. HTML is easy. Read this book in combination with studying the HTML & CSS tutorials on w3schools.

PHP & MySQL:

PHP & MySQL For Dummies, 4th Edition

http://www.amazon.com/PHP-MySQL-Dummies-Janet-Valade/dp/0470527587

This book--well an older edition--I read a year before I got serious about learning to code. I read it and didn't actually code anything i learned, but what it did was plant seeds in my head with regards to what programming is all about and what databases are all about, and how to connect the two. It assumes very little in what you may

already know, and is an excellent start in your journey to becoming a master programmer.

PHP Object-Oriented Solutions

http://www.amazon.com/PHP-Object-Oriented-Solutions-David-Powers/dp/1430210117

This book is where I learned what OOP is. I didn't get the hang of it until reading the following book. Don't worry if you read this and have a hard time with it. This book and the next each have introductory chapters that go over how OOP works. It took me reading basically this book and the next book about the same stuff to get it. This book is a lot less complicated than the following and dives into practical examples & problems, whereas the next is a lot more theoretical.

PHP Objects, Patterns and Practice

http://www.amazon.com/Objects-Patterns-Practice-Experts-Source/dp/143022925X

After reading this book, I basically mastered OOP. It's a very hard book to get through if you're new to OOP, and goes into some very advanced stuff, specifically tons and tons of "design patterns." The design patterns are presented in as basic of a form as possible, but they weren't very practical like examples from the previous book in that you probably will never actually need any of the code used in the book. Either way, this is my favorite Programming of all time because it taught me how to think like a coder and how to solve complex problems with concise refactored solutions. It really showed me what is possible with OOP. You don't know PHP unless you've read this book.

Pro PHP: Patterns, Frameworks, Testing and More

http://www.amazon.com/Pro-PHP-Patterns-Frameworks-Testing/dp/1590598199

I read this book too just to solidify my experience with PHP, and cover all my bases. Check it out. It's optional.

PHP Functions Essential Reference

http://www.amazon.com/Functions-Essential-Reference-Torben-Wilson/dp/073570970X

At some point during my study of PHP I found this book and decided just to learn every PHP function available so that I could better understand the examples in the above books. Start reading this early on, and complete the whole thing. You'll quickly learn patterns in how PHP functions are named, and as a result be able to guess what a function does within the context of the examples in the above books--even if you don't remember precisely what it does.

Agile Web Application Development with Yii 1.1 and PHP5

http://www.amazon.com/Agile-Web-Application-Development-PHP5/dp/1847199585

I read this book in combination with reading the Blog Tutorial and Definitive Guide on YiiFramework.com. When you're done studying all these materials, you'll be amazed with how much power you have. This book isn't hard to read either. You'll love it if you reach this stage!

Javascript & jQuery:

Learning jQuery, Third Edition

http://www.amazon.com/Learning-jQuery-Third-Jonathan-Chaffer/dp/1849516545

jQuery is a framework built on top of the native browser language of Javascript. Usually one would recommend you learn the base language--Javascript--before learning an abstracted framework on top of it--jQuery. However because of the nature of jQuery and how comprehensive it is and because of how quirky Javascript is coming from PHP, I found it best to jump to jQuery and immediately start accomplishing the DOM manipulation tasks I needed. And ultimately because of syntax similarities between PHP and Javascript I was able to get productive in Javascript without studying a single book just on Javascript. One thing I did different

when studying this book from the PHP books is I did every single tutorial as I read it. The reason is because when I learned PHP, I was learning my first real programming language--so it took me a lot of time to just digest things before I could code a single line, which is why I just read PHP book after PHP book before I got started until it all made sense. However, by the time I got to Javascript & jQuery, I understood how programming in general works and found it helpful for memorization purposes to immediately start doing the tutorials.

jQuery 1.3 with PHP

http://www.amazon.com/jQuery-1-3-PHP-Kae-Verens/dp/1847196985

With this book I didn't do all the tutorials like I did with *Learning jQuery*, but what reading this book did for me is taught me precisely how Ajax works and what it's all about. After reading it, coding features that required Ajax using Yii and PHP was obvious and a no-brainer.

JavaScript: The Good Parts

http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742

This book gave me a deep understanding of the Javascript language and what it's truly all about. After reading it, many hours of debugging and head-scratching when coding Javascript & jQuery were removed from my schedule--because I finally learned the quirks of the Javascript language I needed to know.

Pro JavaScript Design Patterns

http://www.amazon.com/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X

Now this book took my Javascript game to the next level, gave me an idea of how jQuery was built, taught me how to do things similar to how you would in a "classical" OOP language like PHP, and completely ended any remaining head-scratching I was having with Javascript, particularly with how "scope" works in Javascript.

Linux:

The Official Ubuntu Server Book, 2nd Edition

http://www.amazon.com/Official-Ubuntu-Server-Book-2nd/dp/0137081332

Note: by the time I read this book I had already learned Linux through blogs on the internet. The best thing I can recommend you do is install Linux on your computer from the Ubuntu website, and start navigating around the command line, practicing Linux commands you learn off the web. Just google "linux tutorials" and you'll be off to a running start. That said, by the time I got proficient in Linux and after I read this book, I felt confident that I really knew what I was doing and had practical solutions for the most common problems you'll face at the command line.

Apache Cookbook: Solutions and Examples for Apache Administrators

http://www.amazon.com/Apache-Cookbook-Solutions-Examples-Administrators/dp/0596529945

This book I treat like a pocket reference and still refer to it often since it's impossible to remember all the different Apache configurations, given how comprehensive this web server application is. I did read it through when I first got it. I kinda skimmed it though--just to get an idea of what is possible. Getting an idea of what is possible without mastering a subject matter is so important in programming because you'll know where to look when you face a challenge that the subject matter can solve.

Pro Bash Programming

http://www.amazon.com/Bash-Programming-Experts-Voice-Linux/dp/1430219971

This is a little advanced for readers of the *FaceySpacey Bible*, but I'm putting it here because it really took my Linux skills to the next level.

NON-TECHNICAL BOOKS:

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385

If you plan to grow a small application into a large company, this book is a must. It's short. Read it.

SEO Book.com

http://www.seobook.com

This obviously isn't a book, but I read their entire site like a book, and its creator, Aaron Wall, expects you to read it like a book. When I was done reading it, I felt I was completely up to speed regarding what SEO is, how search engines work, and practical techniques to get better rankings in search engines.

For daily Startup Wisdom, checkout FaceySpacey.com/blog daily. And don't forget to download the entire No Bullshit FaceySpacey Bible or more individual chapters here: http://www.faceyspacey.com/resources?section=book .

*Thanks again from FaceySpacey and be sure to check out **FaceySpacey.com** often for further knowledge we kick to take your Startup to the Stars!*